NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

EVALUATING CONFIGURATION MANAGEMENT TOOLS FOR HIGH ASSURANCE SOFTWARE DEVELOPMENT PROJECTS

by

Lynzi Ziegenhagen

June 2003

Thesis Advisor:	George Dinolt
Second Reader:	Michael Thompson

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Title (Mix case letters) Evaluating Configuration Management Tools For High Assurance Software Development Projects	**5. FUNDING NUMBERS**
6. AUTHOR(S) Lynzi Ziegenhagen	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE** A

13. ABSTRACT ***(maximum 200 words)***

This thesis establishes a framework for evaluating automated configuration management tools for use in high assurance software development projects and uses the framework to evaluate eight tools. The evaluation framework identifies a dozen feature areas that affect a high assurance project team's ability to achieve its configuration management goals and evaluates the different methods that existing tools use to implement each feature area. Each implementation method is assigned a risk rating that approximates the relative risk that the method adds to the overall configuration management process. The tools with the lowest total ratings minimize risk to high assurance projects.

The results of the evaluation show that although certain tools are less risky to use than other tools for high assurance projects, no tool minimizes risk in all feature areas. Furthermore, none of the existing tools are designed to leverage high assurance environments—i.e. none run on operating systems that have themselves been evaluated as meeting high assurance requirements. Thus, high assurance development projects that want to leverage the benefits of configuration management tools and achieve a sufficiently strong configuration management solution must employ existing tools in a protected environment that specifically addresses the risks created by the tools' implementation methods.

14. SUBJECT TERMS High Assurance, Configuration Management, Computer Software, EAL7	**15. NUMBER OF PAGES** 107
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

EVALUTING CONFIGURATION MANAGEMENT TOOLS FOR HIGH ASSURANCE SOFTWARE DEVELOPMENT PROJECTS

Lynzi Ziegenhagen
Civilian, United States Department of Defense
Undergraduate (B.S.), Stanford University, 1994

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2003

Author: Lynzi Ziegenhagen

Approved by: George Dinolt
Thesis Advisor

Michael Thompson
Second Reader

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis establishes a framework for evaluating automated configuration management tools for use in high assurance software development projects and uses the framework to evaluate eight tools. The evaluation framework identifies a dozen feature areas that affect a high assurance project team's ability to achieve its configuration management goals and evaluates the different methods that existing tools use to implement each feature area. Each implementation method is assigned a risk rating that approximates the relative risk that the method adds to the overall configuration management process. The tools with the lowest total ratings minimize risk to high assurance projects.

The results of the evaluation show that although certain tools are less risky to use than other tools for high assurance projects, no tool minimizes risk in all feature areas. Furthermore, none of the existing tools are designed to leverage high assurance environments—i.e. none run on operating systems that have themselves been evaluated as meeting high assurance requirements. Thus, high assurance development projects that want to leverage the benefits of configuration management tools and achieve a sufficiently strong configuration management solution must employ existing tools in a protected environment that specifically addresses the risks created by the tools' implementation methods.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to acknowledge the National Science Foundation for its financial support of my master's degree through the Scholarship for Service Program.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE OF STUDY

This thesis is part of the Diamond High Assurance Security Program's Trusted Computing Exemplar Project of the Naval Postgraduate School's Center for INFOSEC Studies and Research (CiSR). The Exemplar project will provide an "openly distributed ***worked example*** of how high assurance trusted computing components can be built" [CISR02]. The Exemplar project is a response to the current dearth of high security, high assurance, off-the-shelf products available to protect the National Information Infrastructure. By providing to the broad community a prototype framework for high assurance system development as well as a trusted computing component developed and evaluated under this framework, the Exemplar project will begin to fill the existing void and enable others to more easily pursue trusted computer systems and networks.

A key part of the Exemplar's prototype framework for constructing trusted computing systems and components is the configuration management of specifications, software, tools, processes, etc. Effective configuration management requires the use of an automated configuration management software tool. The goal of this thesis is to both help the Exemplar project and other high assurance projects select the CM tool that best meets their needs, and to help CM tool vendors to better understand how to design their tools to better support high assurance projects.

B. INTRODUCTION TO HIGH ASSURANCE SYSTEMS

1. High Assurance

Many vendors today claim that their systems are "high assurance" or "secure." How do we evaluate such claims? One way to think about such claims is to compare them with the claims a hotel makes about the security of its system for guarding your valuables.

"You can rest assured your jewels are safe with us," the manager insists. But before you will have any assurance in the hotel's system for guarding your jewels, you need to:

1. Understand what the hotel means by "secure;"
2. Evaluate the measures the hotel is taking to provide this security; and
3. Verify the measures yourself.

First, you ask what the hotel means by "secure." What are the threats they are protecting you from? Loss? Theft? Natural Disasters? Damage? The hotel hands you a form detailing their liability. It says that your valuables are guaranteed against theft up to $15,000, but in the event of a natural disaster or fire, they take no responsibility for any damage or loss. The manager points out that other hotels are much less secure – they do not guarantee anything; they just claim you would have a reduced risk of theft if you leave your valuables to the front desk instead of in your hotel room.

Next, you ask what measures the hotel has taken to ensure that the jewels are safe. The hotel tells you that they have a high-quality safe, and the manager and assistant manager on duty carry the only two keys to the safe. Guards monitor the safe via video camera at all times. Before items are placed within the safe, they are attached to a label with the guest's name and signature. The front desk requires two photo IDs in and a matching signature in order to withdraw the jewels from the safe.

Assume you are satisfied with the description of the measures provided, but you want to verify for yourself that the hotel has actually implemented the manager's claims. So you ask if you can see for yourself that the video camera is installed and working. You ask to see the log listing guests' signatures. You check that the manager and assistant manager have keys on them that do open the safe. All seems well. But how can you check that their keys are the only keys to the safe? Perhaps some of the keys to the guest rooms also open the safe. Perhaps they leave a key next to the safe.

Despite these last questions, you may have enough assurance in the hotel's security definition and security measures, given your independent verification, that you would leave your jewels in the safe. But given that you can't verify the non-existence of other keys to the safe, you can't have "high assurance" in the hotel's claims. As elaborated below, an evaluator of a "high assurance" computer system *can* verify the

non-existence of "other keys" (in computer terms, can verify the non-existence of trap-doors).

2. High Assurance in Computer Software Systems

A computer software system's claim to be "secure" is just a claim unless it fulfills the same requirements the hotel fulfilled: the system's makers must provide:

1. A precise definition of "secure;"
2. The measures taken to implement the specific security definition; and
3. A way for third parties to verify that the measures enforce security claims.

Whether and how the system's makers fulfill these requirements determines the degree of assurance you can have in the system's security.

a. Definition

The precise **definition** of "secure" is provided by a security policy. The security policy states the assets (e.g., files) and the threats (e.g., unauthorized access) against which the system protects its assets. In high assurance systems, the definition of secure includes:

1. Only and exactly the functionality as defined. That is, the system has not been subverted with unspecified functionality (e.g., there are no extra keys to the safe in the hotel analogy). System subversion involves the "clandestine and methodical undermining of a system by planting artifices (trap doors) in it that bypass its security controls" [ANDR02].
2. Bounded information flow (per the security policy)

b. Measures

The **measures** that secure system development efforts take to ensure that the system developed enforces the given security definition will vary based on the level of assurance the system claims to provide. For high assurance systems, there are five key measures:

1. Proving that the security policy is clear and consistent. To do this, the policy is translated into a formal security model whose claims and consistency can be proved using mathematical proving tools.

2. A system architecture and design that can be evaluated, e.g., through the use of abstraction, layering and information hiding with significant system engineering directed toward minimizing the complexity of the protection mechanisms and excluding from the security perimeter modules that are not protection-critical. A descriptive top-level specification (DLTS) completely and accurately describes the protection mechanisms.
3. A Formal Top Level Specification (FTLS) with precise syntax and well-defined semantics, which is shown to be a complete and accurate representation of the security perimeter and to be internally consistent.
4. Code correspondence that explicitly maps each line of code back to the FTLS to ensure that the implementation is complete, error-free and includes no additional, unspecified functionality.
5. Testing results from a test plan driven by FTLS to verify presence of security functionality and absence of functionality that would violate the desired security properties of the system.

c. Verification

Systems may provide various types of verification of their security claims. In high assurance systems, the artifacts listed above provide the **verification** that the system is "secure" (as defined by the security policy model). These artifacts (security model, proof of the model, DTLS, FTLS, proof of the FTLS, code, FTLS-to-code correspondence, implementation documentation, test plans, test results) are available for anyone to independently review. Anyone can regenerate the proofs and verify the security claims, including the absence of subversion.

3. Official Definitions of High Assurance

There are two primary sources that the computer industry relies upon when it comes to High Assurance Systems. The first one is the United States Department of Defense's (DoD's) 1985 Directive to use *Trusted Computer System Evaluation Criteria*, better known as the Orange Book [DODD85]. The Orange Book's purpose was to provide security criteria and technical evaluation methodologies to support the DoD's systems security policy and the DoD's evaluation and approval/accreditation responsibilities [ORNG85]. The Orange Book established six hierarchical classes and each class' requirements, both in terms of security features and assurance. The most

trusted, highest assurance class was known as A1. Generally speaking, when people talked about “high assurance” or “trusted” systems, they were referring to systems which could be evaluated at A1.

Systems that were evaluated at A1 (or equivalent level) in the past 20+ years include SCOMP, Gemini Trusted Network Processor, and Boeing MLS LAN products [EVAL03]. Note that although evaluations at the A1 level were system evaluations, not all parts of the system had to fulfill the same requirements. The parts that comprised the Trusted Computing Base (TCB) were wholly responsible for maintaining the system’s security properties. Designers had to demonstrate that the non-TCB components of the system could not violate the policies enforced by the TCB.

Though many in the security field still refer to the Orange book because multiple systems were evaluated using its criteria and because it directly addressed subversion, officially the Orange Book was canceled by the DoD in October 2002, by DoD Directive 8500.1, "Information Assurance (IA)” [DODD02]. The new DoD standard is the Common Criteria, an international effort to develop criteria for evaluating information technology security. The Common Criteria provides a scale for rating the assurance level of a system based on defined characteristics. The Criteria’s highest assurance level is “Evaluation Assurance Level 7: Formally Verified Design and Tested” (EAL7). Note that EAL7 itself does not include any specific functional features, as the Orange Book’s A1 did; it is solely focused on assurance.

C. INTRODUCTION TO CONFIGURATION MANAGEMENT (“CM”)

Software development efforts are marked by a number of challenges that can easily create chaos, including:

- High complexity
- Large teams
- Widely dispersed teams
- Developers working in parallel
- Changing requirements
- Multiple versions for different markets or customers
- Pressure to meet customers’ needs quickly

In such a chaotic environment, critical questions such as who made recent changes (and why and with whose approval), when some piece of code broke, what code a specific customer is using, and which components are related, often cannot be answered easily [BERC03].

CM attempts to prevent the chaos. CM is the disciplined approach of controlling changes in a large and complex system throughout its life cycle [IRCM02]. CM can be used to control any type of system development, but this thesis focuses on CM for software system development. CM in software development is the "disciplined approach to managing the evolution of the software development and maintenance practices" [DART00]. In a formal sense, the "objective of CM is to ensure a systematic and traceable development process, so that at all times a system is in a well-defined state with accurate specifications and verified quality attributes" [IRCM02]. CM should validate and maintain the system's integrity by ensuring that the systems' objects are the appropriate ones.

But how is CM done? How does it prevent chaos? CM identifies all items involved in the development process, controls these items through any and all changes, accounts for the items' statuses, and audits items to ensure that any composite items (i.e. "configurations") are a valid, consistent set of components [DART00]. CM is like the hotel's safe and the procedures for adding and removing items from the safe that keep guests' valuables secure.

> "CM is pervasive across the software development and maintenance life cycles. It is the core support system that enables safe and efficient development and maintenance" [DART00].

CM provides many benefits to a development effort. At a high level, CM serves as a mechanism for communication, change management, and reproducibility [BERC03]. In addition to "control over everything" [DART00], CM provides numerous business and technical benefits (selected from [DART00]). See Table 1. Note that some of these benefits require the use of an automated CM tool.

Business Benefits of CM	Technical Benefits of CM
• Insurance against the unknown • Very easy audits • Foundation for process and quality improvement • Eliminate avoidable mistakes • Fewer bugs in released product • Automatic quality control • Teamwork optimization • More product lines/versions possible	• All objects versioned • Failure recovery, rollback support • Repeatability of all steps • Faster change cycles • Queries • Audit Log • Process enforcement • Enable standards certification • Minimal change complexity • Change propagation • Parallel development

Table 1. CM Benefits

D. HISTORICAL CONTEXT OF CM IN HIGH ASSURANCE SYSTEM DEVELOPMENT

The definitions of CM given in Section C.1 are from books and articles targeted for the general development market; the sources give scant if any mention of assurance or security issues. For example, Dart's one reference to security is, "Programmers must be very cognizant now of security issues since, in theory, the world could try and hack into the Web system" [DART00].

High assurance efforts tend to have less chaos than typical software efforts, since the software's complexity must be relatively low and its teams small and centralized. So one might imagine that high assurance efforts need CM less than other efforts. But the purpose of CM is not just to tame the chaos, but to "Ensure that nothing stray enters the system by accident or maliciously" [DART00] (i.e. that the system is not subverted). Unlike most software, which is judged primarily by its functionality, high assurance software is judged firstly by its assurance; it isn't high assurance software if something stray does enter the system. In addition to keeping out stray items, high assurance efforts need to provide a concise and complete definition of the protection mechanism: to clearly identify what is part of the mechanism and what is outside of it. Since CM helps to prevent subversion and to identify what is and is not part of the protection mechanism, CM plays a pivotal role in the success of high assurance efforts.

1. How the Relationship Began

CM has its roots in manufacturing, not software. As products got bigger and more complex in the middle of the twentieth century, manufacturers became increasingly unable to manage product development cost effectively without formal controls. Thus in 1962, the American Air Force, in response to control and communication problems that occurred during the design of its jet aircraft, developed and published a standard for CM (AFSCM 375-1) [LEON00].

Software development teams quickly identified CM's usefulness for software development. In the early days of CM, before the availability of sophisticated automated CM tools, CM processes were tedious and time consuming. This limited the use of CM to development efforts where the cost of time and organization was worth the benefits of reduced errors and improved communication, namely in large and complex systems and systems where an error might have catastrophic consequences to financial assets, the environment, human life, or national security (i.e. high assurance systems) such as SACDIN, the "primary network for the transmission of Emergency Action Messages (EAMs) to the warfighting commanders in the field" [FOAS03].

High assurance systems also valued configuration management because it provided another defense against system subversion through tighter control of changes and increased visibility to changes.

Since 1962, the US government and international bodies have established a multitude of standards for CM. The standards explicitly define what CM plans, procedures, and policies are required to provide certain levels of assurance. For example, RTCA DO-178B, which was created in 1992, "defines a set of objectives that are recommended to establish assurance that airborne software has the integrity needed for use in a safety-related application" [RTCA03] and includes CM objectives. However, the most well known standards for high assurance software system development today are the Orange Book and the Common Criteria.

2. CM According to the Orange Book

The US Department of Defense established specific CM requirements for high assurance ("A1") systems, as detailed in the Orange Book in Section 4.1.3.2.3. Specifically, the DoD required a CM system to be in place during the entire life cycle of the system which maintains the system's consistency and from which one can generate a new version of the system from the source code and compare it to a previous one.

Besides the standard CM requirements, the Orange Book included a number of requirements unique to high assurance systems. The focus of CM, according to the Orange Book, is to ensure that the hardware and software are protected "against unauthorized changes that could cause protection mechanisms to malfunction or be bypassed completely" [ORNG85 5.3.3]. See Table 2 for some of the A1 requirements. Note that the Trusted Computing Base is considered the part of the system that is being evaluated for high assurance. It is typically just a subset of the full system.

CM for Standard Systems	**CM For High Assurance Systems**
CM used only during Implementation Phase.	CM used during "entire life-cycle" [ORNG85 4.1.3.2.3].
Track only the software being created.	Track "all security-relevant hardware, firmware and software" [ORNG85 4.1.3.2.3].
Track only the code and tests.	Track "formal model, the descriptive and formal top-level specifications, other design data, implementation documentation, source code, the running version of the object code, and test fixtures and documentation" [ORNG85 4.1.3.2.3].
Goal of using CM is to minimize chaos.	Goal of CM is to provide assurance that the correct implementation and operation of the policy exists throughout the system's life cycle. Or in other words, "Configuration management provides assurance that additions, deletions, or changes made to the Trusted Computing Base do not compromise the trust of the originally evaluated system" [CMTS88].

CM for Standard Systems	CM For High Assurance Systems
Changes need to be tracked because they might interfere with others' work.	"Configuration items need to be individually controlled because any change to a configuration item may have some effect upon the properties of the system or the security policy of the Trusted Computing Base" [CMTS88].

Table 2. CM for Standard vs. High Assurance Systems

3. CM According to the Common Criteria

The Common Criteria requirements for CM are similar in focus to those of the Orange Book, but they are more specific and they explicitly require the use of an automated CM tool. The Common Criteria divides the requirements into three areas: automation requirements, capabilities or characteristics of the CM system, and scope of the system that needs to be controlled by CM. Many of the requirements are open to interpretation by the evaluators. The key requirements for the EAL7 level are summarized below [COMC99].

a. Automation

The system needs an automated way to: (1) ensure only authorized changes are made, (2) generate system (i.e. create binary files from code), (3) ascertain changes between different versions, and (4) identify all items affected by a modification. Furthermore, the CM Plan needs to describe the automated tools and how they are used.

b. Capabilities

Each version, as well as all of its components and items, must have a unique, documented ID. The CM plan must describe how the CM tool is used and include an acceptance plan describing

- Procedures to accept new and modified items and
- Procedures describing how CM is applied in manufacturing process.

Evidence must be provided that the CM tool is operating as described in the CM Plan, that the system has measures to ensure that only authorized changes are made, and that all items are being "effectively maintained" under CM [COMC99 ACM_CAP.3.9C]. The CM Documentation must describe all security measures, show how integration procedures ensure correct and authorized generation, and justify that the acceptance procedures provide adequate and appropriate review of changes to all configuration items.

c. Scope

CM documentation must describe how the CM tool tracks the following: implementation representation, security flaws, software tools, and documentation of design, test, user, and administration.

Interestingly, the Common Criteria's EAL7 is considerably less explicit than is the Orange book regarded two requirements that help address subversion: protecting the integrity of items in the repository and protecting the integrity of the CM tool itself.

4. The Relationship Today

The relationship between CM and high assurance systems today is difficult to determine, given the limited number of high assurance systems being developed. The development teams for A1 projects used primarily manual CM procedures. Current high assurance efforts are primarily in the air vehicle area, including Lockheed Martin's Advanced Tactical Fighter. Several other systems have been developed using RTCA DO-178B standards, the standard for airworthiness in the U.S., including the Boeing 777 and TCAS (Traffic Alert/Collision Avoidance System).[1]

As stated above, the high assurance evaluation is of the protection mechanisms that enforce the security policy. There are typically non-critical sections of an overall product that are not evaluated to meet high assurance requirements.

---

[1] Interestingly, the paper describing the independent verification of TCAS describes the high level CM process they used, but the only automated tool they mention is a tracking database for formal discrepancy reports [TCAS99].

E. HISTORICAL CONTEXT OF AUTOMATED CM TOOLS

1. CM Before Automated Tools

CM initially consisted of manual processes – lots of tedious, detailed, manual processes. Too many manual processes, as anyone who has had to follow such processes can attest, often cause people to make mistakes or simply skip a step out of frustration or perceived need [BERC03]. The Common Criteria agrees that automated systems are generally superior to their manual counterparts. “While both automated and manual CM systems can be bypassed, ignored, or prove insufficient to prevent unauthorized modification, automated systems are less susceptible to human error or negligence” [COMC99].

Furthermore, before efficient automated tools were available, organizations saw CM as useful primarily just before release [DART00]. CM wasn’t part of a developer’s daily work. Instead, once a developer had a piece of code working, he or she would “toss it over the wall” to the CM librarian. The librarian's concern was on "control, precision, completeness and timing," while programmers' concerns were on creating and fixing code as quickly as possible [DART00]. This difference in focus inevitably led to conflicts.

2. The CM Tool Evolution and Revolution

The earliest automated tools used by CM practitioners were databases that allowed CM librarians to track items, do basic querying, and enforce basic access control. Even as late as 1988, the tool offerings were quite basic, as evidenced by two of the popular tools, UNIX (1) SCCS and VAX DEC/CMS [CMTS88]. Both tools were typically “controlled” by one person (i.e. the librarian). A guide explaining how to use them suggests that they didn’t even provide an automated way to uniquely identify items [CMTS88].

The CM software tools available today are of an entirely different class. As recently as 2000, the market had an annual growth rate of more than 20% [RCM02]**;** dozens of CM products are currently available. CM tools are no longer relegated to the release stage or separated from the developers’ daily work. All work products are under

CM at all times: "everyone involved in the software development and maintenance life cycle can be empowered to do their work in a CM-controlled environment and with independence and integrity" [DART00]. Though CM tools have traditionally focused on the implementation phase of the development process (code, test, build), recent versions of the tools allow you to include pre- and post-implementation processes and artifacts (e.g., requirements, design, deployment, configuration) [LEON00]. Supporting the entire lifecycle through CM is an active area of research [IRCM02].

The benefits of using an automated CM tool are not just automation of repetitive tasks and error reduction. Automated tools also reduce development time, increase business agility, enable organizations to integrate information and analyze it effectively [LEON00].

3. What Today's Tools Can Do

The range of products that call themselves CM tools includes those that only perform basic version control to those that provides integrated, full process management. Full process management tools include features to support CM's primary functional areas [DART00, LEON00].

- Version and configuration control
- Change management
- Configuration item structuring
- Construction of configurations
- Teamwork support
- Process/Promotion management
- Auditing
- Status reporting
- Access and security

See Chapter IV, "CM Tool Feature Area Analysis," for a detailed discussion of the features of modern CM tools and Appendix A for a detailed description of the tools evaluated in this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

II. THEORY FOR USING CM TOOLS IN HIGH ASSURANCE PROJECTS

A. CM'S DUAL ROLE IN SYSTEM DEVELOPMENT

Implementing CM, even with today's automated tools, is a significant effort. Yet virtually all system development projects do implement some form of CM [LEON00]. Why? For the same reasons that the hotel buys a safe and establishes procedures for using it as part of the hotel's valuables safeguarding service: first, the hotel management believes that doing so is the only way to deliver a high quality service; second, they use a safe because they want to be able to tell their guests that they use a safe. They know that guests will not be impressed with a service that leaves the storage location of valuables to the discretion of the front desk staff.

Similarly, software development vendors pursue CM because they believe it improves the overall "quality" of their product and because they want to impress external parties such as customers, partners, and evaluators. Vendors know that being able to demonstrate their use of CM increases others' perception of the quality of their product. Many industry standards for software development include CM requirements (e.g., Capability Maturity Model for Software [CMMS95]).

Vendors developing high assurance software products are especially concerned about both the quality of their products and external parties' opinions of their processes. The evidence created during the high assurance process demonstrates that the system maintains its stated security properties and thus is a quality product. If the evidence is not created, because of a failure of CM or of another part of the development process, the system is considered a failure, even if it performs other stated functions adequately.

The CM requirements for "impressing" external parties are explicitly defined in existing assurance standards such as the Orange Book and the Common Criteria. The standards are used by evaluators to determine the level of assurance that the system provides and by external parties to understand the product's assurance level. Because CM provides a level of control around the development process, CM is one of the primary

sources of assurance for development efforts. Each increased level of assurance, up to and including "high assurance" systems, those that are the focus of this thesis, requires more extensive CM.

B. IS CM RELEVANT IN HIGH ASSURANCE SYSTEM DEVELOPMENT?

The development process for high assurance systems creates a trail of evidence that is supposed to permit "after-the-fact" evaluation. If the evidence is necessary and sufficient to permit "after-the-fact" evaluation, then why are CM practices—which are primarily about controlling change during the development process—required? Do evaluators not believe that the evidence is necessary and sufficient? Or are they requiring CM, much the way that guests require hotels to have safes, because evaluators know CM is the best way for a vendor to be able to successfully create the evidence?

Let's address each of these questions in turn. Why might evaluators not consider the evidence to be necessary and sufficient? Some may be concerned about the two links in the evidence trail that are only informally proven: the link between the security model and the written security policy and the link between the code and the FTLS. But if evaluators are looking to bolster the product's assurance level with CM, their efforts are fundamentally misguided. CM consists of human processes and procedures layered on top of complex, low assurance software running on low assurance operating systems. Expecting processes, procedures, and low assurance software to provide assurance that evaluators will not give to semi-formal proofs is like relying on a hotel's excellent management of their safe's keys to provide assurance when the safe itself is made of cardboard.

Is CM required because evaluators believe that CM is the best way for a vendor to be able to successfully create the evidence? Clearly, the internal reason to use CM—to improve the overall "quality" of the product—is very compelling. Any vendor attempting to develop a high assurance system without CM (just as any hotel attempting to protect guests valuables without using a safe) would find creating the requisite evidence very difficult, perhaps impossible. But were a vendor able to create the evidence without CM, and the evidence verified the security properties of the system, it seems ridiculous for an external party to reject the system because the vendor's CM processes were not up to

standards. If you received your jewels back after storing them with the hotel for several days, would you then ask the hotel to prove to you that its safe is used appropriately? No. You would happily put on your jewels and go about your merry way.

So should CM still be required in high assurance evaluations? The answer is yes, and the reasons are practical ones based on the limitations of the evaluation process. Though the evidence is *theoretically* necessary and sufficient, in practice, verifying the evidence created by a high assurance system development effort is a Herculean task. Why is it so difficult? First of all, the size and complexity of the evidence created by typical high assurance systems efforts is so great that one person cannot comprehend it thoroughly enough to have complete confidence in its correctness. Secondly, even if one person were able to understand all the evidence, the amount of time it would take her to do this would be impractical. For example, KSOS had 50,000 lines of code in the kernel and another 500,000 lines performing security-related functionality [RADL03]. Thoroughly verifying the code correspondence for such a system could take more time than is practical in today's fast-paced technology market. By the time the evaluators delivered their decision, the product could be obsolete.

Furthermore, software products tend to change over time because of new requirements. Even if a product were to be exhaustively evaluated for release 1.0, exhaustively re-evaluating the entire evidence after a small change to the design and code would be impractical. It is more practical to simply evaluate the small changes to ensure that "the additions, deletions, or changes made to the Trusted Computing Base do not compromise the trust of the originally evaluated system" [CMTS88]. But in order to give the new version of the product the same rating as the previous version, the evaluators also have to have assurance that the previous version has not been changed—i.e. that it is the same product that was evaluated previously.

C. CM'S PROPER ROLE IN HIGH ASSURANCE SYSTEM DEVELOPMENT

Since evaluators' verification of the evidence might be limited to rerunning the proofs and doing spot checks of the other evidence, evaluators look for other signs that the vendor created the evidence correctly. As described above, effective CM gives

vendors the best tool for creating the appropriate evidence. Though CM does not *increase* the assurance level beyond what the evidence provides, *lack* of appropriate CM reduces the assurance level. Similarly, knowing that a hotel has no safe or no procedures for using the safe reduces your confidence that the jewels returned to you were your real jewels.

In addition to enabling the creation of the evidence, CM plays three other practical roles once the evidence and the product is created and evaluated.

- Identify the evaluated product and its evidence. Which components belong to the evaluated product? There may be several sets of evidence that appear consistent; how does one know which set was evaluated?
- Protect the evaluated product and its evidence. Once the product has been evaluated, it must be protected against modification or corruption so that it can be distributed and so that it can be compared to future versions.
- Compare a distributed version to the evaluated product to ensure equivalence. Once the evaluated product has been distributed, the recipients need a way to verify that their version of the product is the version evaluated and thus has the reported assurance level.

These roles do not increase the assurance of the product; they are practical roles that help the vendor ensure that its evaluated product does not become corrupted and can be used by others.

CM is thus an important part of high assurance system development, both for internal purpose of enabling the creation of evidence and for the external purpose of impressing external parties such as customers and evaluators.

D. CM TOOLS' ROLE TODAY IN HIGH ASSURANCE EFFORTS

As discussed earlier, the role of automated tools in high assurance CM was initially limited. People—usually with a security clearance—were responsible for enforcing the policies and establishing and following manual procedures. Automated tools assisted in limited, isolated areas, such as comparing two documents, physically storing the files under configuration, and tracking identification numbers of items.

Today's configuration management tools are capable of automating most CM procedures and enforcing a significant number of CM policies. Of course, just because

something can be automated does not mean that it should be automated. What are the benefits of CM automation to high assurance efforts and what are the drawbacks? How should one determine which aspects of CM to automate and which to keep manual? Automation can take different forms. Which implementations are most beneficial to high assurance efforts?

At first glance, automation seems to be beneficial in most respects, for it provides standardization and auditability, and reduces errors caused by tedious, manual procedures.

However, all of the existing CM tools have one significant flaw: they are themselves low-assurance software running on low-assurance operating systems. The tools may have features that enforce CM policies and automate CM procedures, but the tools' low-assurance environments mean that their features cannot be trusted to enforce the policies correctly all of the time, nor to provide sufficient resistance to an attack by a malicious user. Thus, the tools cannot be fully trusted to create the evidence correctly, nor to protect, identify and compare the product once created.

So does one throw out the tools and return to old-style CM using single-task tools on physically protected machines administered by cleared personnel augmented by manual, paper-based procedures? Not completely. One way to get the benefits of the existing CM tools while taking into account the low-assurance risks is to establish both "Everyday CM" that uses a modern tool and a "Trusted CM" which uses some of the traditional procedures and serves as the auditable CM process.

1. "Everyday CM"

The guiding principle of "Everyday CM" is to take advantage of the modern tools while limiting the opportunities for errors and subversion. "Everyday CM" is not just the engineering system's CM. It consists of four key principles:

a. Select the Best CM Tool

Select the CM tool that best implements your CM procedures and enforces your CM policies. Chapter IV provides an overview of the feature areas of existing tools

that are relevant to high assurance efforts; Chapter V evaluates existing tools according to the different implementations of the features.

b. Control the Tool's Environment

Setup, administer, and use the selected CM tool in the way that maximizes the likelihood of your team to successfully create the high assurance evidence. Because existing tools operate in low assurance environments, they lack effective mechanisms for fighting subversion. Thus, high assurance efforts using CM tools need to reduce the threat of subversion by providing other types of security around the use of the CM tool, like those used in past evaluated high assurance systems, including:

- Physical security
- Restrict access to personnel with a need to access the information or with an established level of trust (i.e. cleared or investigated personnel)
- Maintenance of separate CM systems for high assurance development (i.e. work on the protection mechanisms) and for other development
- Separation of the network on which high assurance work is being done from other networks, including, of course, the Internet.

c. Treat "Everyday CM" As If It Were *"Trusted CM"*

"Everyday CM" is not "just" the engineers' CM where engineers are free to use whatever processes they please to deliver the goods to the "Trusted CM" team. Instead, the CM requirements in EAL7 should serve as a guide for "Everyday CM." There should be a CM plan[2], procedures to accept new and modified items, and measures to ensure that only authorized changes are made [COMC99]. Two important caveats are required, however.

First, there are items that are under the control of the CM tool, but not "under CM" in the sense that the CM Plan's policies and procedures apply to them. These

---

[2] The creation and implementation of the CM Plan are critical to the success of the CM effort, but are not discussed here. A useful reference for a CM plan for a high assurance effort is the Final Evaluation Report for the Gemini Trusted Network Processor [GTNP95]. For an excellent step-by-step guide to implementing automated tools, see [DART00].

items are the ones in individual developers' workspaces and in the first few merges into the software version tree that represent initial testing and integration. Here, engineers can use whatever processes they please. The "procedures to accept new and modified items" only apply to configurations that are ready to be verified more formally, typically the items in branches close to the trunk of the tree.[3]

The second caveat is in the enforcement that "only authorized changes are made." The controlled environment should provide adequate protection against outsider subversion, but does not protect against a malicious insider. Since the tools do not provide reliable protecting against malicious insiders either, the confidence one can have in the enforcement should be measured.

d. Deliver to "Trusted CM"

At significant milestones, deliver the baseline developed by "Everyday CM" along with the baseline history to the "Trusted CM" team for verification and safeguarding.

The hope is that "Everyday CM," by following EAL7 requirements, will enable the developers to create the evidence successfully most—if not all—of the time.

2. "Trusted CM"

To counter the risk of a malicious insider who manages to overcome the environmental and software controls of "Everyday CM," "Trusted CM" is required. The "Trusted CM" environment is more tightly controlled, has additional physical security, is accessable to a smaller group of people, and is completely physical separated from other computer systems. The only functions that the "Trusted CM" team performs are:

- Verify and accept a baseline
- Compare the new baselines with previous baselines to identify changes and verify that they are authorized and appropriate by using the documentation provided by the "Everyday CM" team.
- Accept verified changes into the baseline under "Trusted CM"

---

[3] Not all tools use this traditional branching terminology and structure, but the concepts are similar. See section IV. I for a discussion of how tools can be used to support different lifecycles.

- Be able to identify any specific baseline (e.g., the evaluated product and its evidence)
- Protect the baseline from modification
- Support others' ability to compare a distributed version to the evaluated product to ensure equivalence

Note that the last three functions are exactly the last three roles that CM plays in configuration management (see Section II.C).

Whether "Everyday CM" fulfills EAL7's requirements is a matter of debate, since the interpretation of "adequate measures to ensure that only authorized changes are made" is open to interpretation. To provide more assurance, the "Trusted CM" is the CM that the evaluators will use for their evaluation; thus it too has to fulfill the requirements of the Common Criteria's EAL7. When selecting an automated CM tool for the "Trusted CM" team, the requirements that the automated tool must support are very basic:

- Ensure only authorized changes are made,
- Generate system,
- Ascertain changes between different versions, and
- Identify all items affected by a modification.

The "Trusted CM" establishes and maintains the baselines submitted by "Everyday CM." If the "Trusted CM" team discovers some problem with the baseline sent to them, they will not accept it into their system and will report the issues to the "Everyday CM" team, which is responsible for fixing the problem.

3. Future Work

Future work is needed to precisely determine the appropriate interaction between "Everyday CM" and "Trusted CM." Among the questions that need to be answered are:

- In what format and through what method should the Everyday CM team provide the baseline to the Trusted CM team?
- What metadata needs to be delivered to the "Trusted CM" team in order for them to do their job? In what format should the metadata be delivered? What history is kept with the baseline and what is not necessary for the purposes of the "Trusted CM" team?

- How involved do the “Trusted CM” team members need to be in the “Everyday CM” to be able to accurately verify the baseline? For example, how would they detect a phony change authorization? One could argue that the “Trusted CM” team should participate on the “Everyday CM” team and have total control of the “verified” branches.
- How do TCB development and non-TCB development get integrated?

THIS PAGE INTENTIONALLY LEFT BLANK

III. METHOD FOR CM TOOL EVALUATION

A. EVALUATING TOOLS FOR "EVERYDAY CM"

This thesis focuses on evaluating CM tools for use in "Everyday CM." "Everyday CM" supports the day-to-day development as well as the initial stages of verification; its requirements are significant. Because the "Trusted CM" team is small and trusted, and because its activity is limited to infrequent updates of major baseline releases, the benefits of modern CM tools are not as significant as they are for "Everyday CM." Thus any set of tools that enables the "Trusted CM" team to fulfill the basic Common Criteria requirements in accordance with the CM plan would be sufficient. In practice, the "Trusted CM" team may decide to use the same tool used by "Everyday CM" in order to facilitate importing and exporting.

B. FROM CM ROLES TO CM GOALS

The four roles CM must play in high assurance efforts, detailed above in Section II.C, are:

- Enable evidence creation
- Identify the product and its evidence
- Protect the product and its evidence
- Ensure distributed version is evaluated product

The first role is the role fulfilled by "Everyday CM". "Trusted CM" fulfills the remaining three roles. Table 3 below breaks down these roles by specific CM goals, most of which correspond to existing requirements from the Orange Book and/or the Common Criteria.

Each goal has a number of threats to it, which are included in the table as well. Most of the threats can be characterized as subversion threats, thus their risk can be minimized by the environmental protections described above.

		CM Roles				
Goals	**Threats**	Create	Protect	ID	Ensure	Standards Ref.
1. Control all evidence and all tools	• 3rd party tools used by vendor could be compromised before being put under CM • 3rd party tools could have trapdoors • User could use a tool (e.g., prover) that is not under CM to get the results he wants and then try to put fake results under CM	X	X			[COM C99], [ORN G85]
2. Separation of privilege with change and commit	• User mechanism could be compromised either by guessing or brute-forcing a password or bypassing access control mechanism	X				[COM C99]
3. Maintain consistency of evidence mapping	• Documents could be added without their appropriate upstream counterparts (e.g., proof of security model without the security model) • Documents could be added that don't correspond to existing ones • Person(s) responsible for managing the above process could make errors, unintentional or malicious • Non-atomic commits that could lead to partial configurations	X	X			[ORN G85]
4. Only Authorized Changes	• *See threats under Goals 2 and 3.*	X				[COM C99]
5. Identify all components affected by a change	•	X				[COM C99]
6. Implement CM plan	• Person(s) do not follow manual parts of plan appropriately	X	X	X	X	[COM C99], [ORN G85]

		CM Roles				
Goals	**Threats**	Create	Protect	ID	Ensure	Standards Ref.
7. Protect integrity of items in repository	• Some data is randomly corrupted (i.e. availability and integrity issue) • Attacker is able to modify data in a desired way by going outside of the program • All powerful admin can modify items that are supposed to be immutable (such as committed items, configuration contents, or logs)	X	X			[ORN G85]
8. Protect integrity of CM tool itself	• Unauthorized user gets access to machine directly • Unauthorized user gets access to machine remotely • User with access to system is able to damage or replace CM tool version	X	X			[ORN G85]
9. Clearly identify what is the TOE	• If files were not tightly linked together by CM tool, user could substitute a file w/o detection • If user were to change labels, he could mislead other users and prevent the TOE from being identified.			X		[COM C99]
10. Generate TOE	•			X		[COM C99], [ORN G85]
11. Compare Versions	• If compare doesn't work correctly, might not be able to identify unauthorized changes that have been made	X			X	[COM C99], [ORN G85]
12. TOE and all of its components needs unique identifier	• Names could be changed intentionally to confuse users (e.g., change 'Release_2.0_buggy' to 'Release_2.0_stable').	X		X	X	[COM C99]

Table 3. CM Roles, Goals, and Threats

C. FEATURE AREAS RELEVANT TO ACHIEVING CM GOALS

CM tools represent not just a wide range of functionality, but also a wide range of implementation methods for each feature areas. Many feature areas and their implementation choices have a significant impact on a tool's ability to support the achievement of the CM goals defined above. The key feature areas that can affect CM goals are listed in Table 4.

Feature Areas	**Goals Affected**
A. Repository Architecture	Goal 7: Protect integrity of items in repository
B. Repository Structure	Goal 1: Control all evidence and tools Goal 3: Maintain consistency of evidence mapping Goal 7: Protect integrity of items in repository
C. User Authentication	Goal 4: Only authorized changes
D. Access Control Granularity	Goal 4: Only authorized changes Goal 2: Separation of privilege with change and commit.
E. Storage Of Access Control Information	Goal 4: Only authorized changes
F. Configuration Definition And Enforcement	Goal 3: Maintain consistency of evidence mapping
G. Making History Immutable	Goal 3: Maintain consistency of evidence mapping Goal 4: Only authorized changes Goal 8: Protect integrity of items in repository.
H. Change Transaction Atomicity	Goal 3: Maintain consistency of evidence mapping Goal 4: Only authorized changes Goal 5: Identify all components affected by a change
I. Lifecycle Support	Goal 6: Implement the CM Plan
J. Export/Import	Goal 6: Implement the CM Plan
K. Threaded Discussions	Goal 6: Implement the CM Plan
L. Integrity Verification	Goal 7: Protect integrity of items in repository

Table 4. Key CM Feature Areas and CM Goals Affected By Each

D. SELECTION OF CM TOOLS FOR EVALUATION

There are several dozen CM tools on the market today; analyzing them all was beyond the scope of this thesis. The author's goals were to select a set of tools that represented most of the functionality in the market today. Tools that appeared to be very similar in the key feature areas to tools already selected were not evaluated, since an evaluation would not add value to our analysis. The absence of a tool from the author's

analysis does not mean that the tool is any more or less suitable for use by a high assurance development project. To determine a tool's suitability, one can simply identify the tool's implementation method for each of the key feature areas and add up how well these methods support high assurance CM goals.

We used the following general criteria to select tools for evaluation:

1. Market Share

We were interested in evaluating the dominant market players given their influence on how software development is performed in the industry. Rational's ClearCase dominates the market. MERANT is the next largest player, with several offerings including Merant Dimensions. See Figure 1 below. Note that most companies offer a range of CM tools and the chart does not include the breakdown by tool type. Furthermore, keep in mind that the market is dynamic and undergoing consolidation; the chart below represents merely a snapshat in the CM tool history.

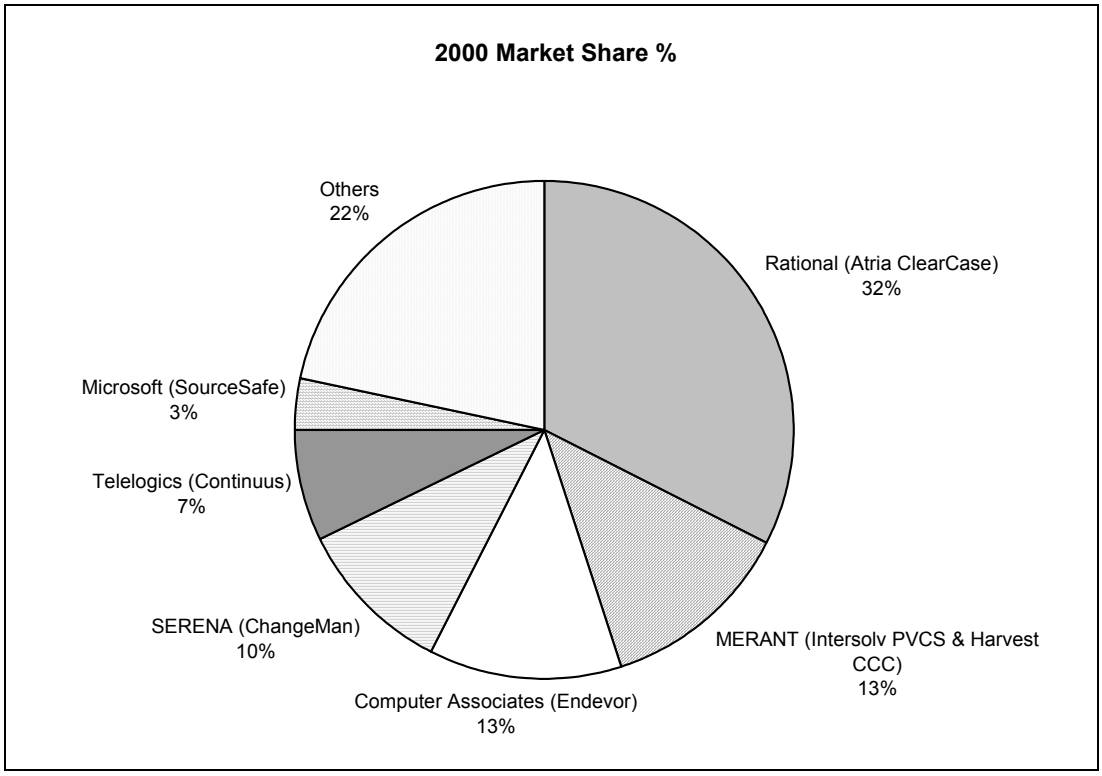

Figure 1. CM Tool Market Share, 2000 [IDC 2000 as reported in IRCM02]

2. Historical Roots of the Product

Some of the tools on the market today, such as Merant's Dimensions, have evolved from older tools. Other CM tools, such as AccuRev, were created from scratch within the past five years. The author selected both types of tools, assuming that the designers of newer tools have learned from the mistakes of earlier tools and designed their tools differently.

3. Range of Functionality

All CM tools provide automated version-control functionality, but only a subset of them claims to provide full process and lifecycle management. Both types add interesting aspects to the evaluation.

4. Open Source

Because of the growing popularity of open source software, the author thought it important to include the most popular open source CM tool, CVS.

5. High Assurance Claims

One tool (OpenCM) claims to be designed specifically for high assurance development projects, and thus is clearly a good candidate for this evaluation, even though it is only in alpha.

6. Unique Features

Other tools with one or two unique features were also selected. For example, BitKeeper was added because of its unique repository architecture (distributed, peer-to-peer); StarTeam was added because it supports threaded conversations and network encryption.

E. GATHERING DATA ON CM TOOLS

To gather data on the selected CM tools, the author used a number of methods. She reviewed written documentation including vendor-created white papers, marketing materials, web sites, and manuals. Where possible, she arranged demonstrations and questioned sales personnel about the features and actual users about their experiences using the tools. Appendix A details the sources for each tool evaluated.

IV. CM TOOL FEATURE AREA ANALYSIS

This section covers in detail each of the functionality areas identified in Section III.C, "Feature Areas Relevant To Achieving CM Goals." Each implementation method of each area is described, along with its advantages and disadvantages for a vendor attempting to create high assurance software. Many of the disadvantages listed below can be mitigated by using the CM tool in a protected environment; in fact, the high assurance team should design the environment to protect against precisely the disadvantages described below.

A. REPOSITORY ARCHITECTURE

The repository is the name given to the collection of files under CM. The architecture of the repository impacts how well the tool is able to fulfill Goal 7 ("Protect integrity of items in repository").

1. One Central Repository Plus User Workspaces

a. Description

Most CM tools have one central repository. Individual team members have their own workspaces they use for development or editing. A member copies objects (code, specs, etc.) from the repository to their workspace, makes changes within their workspace, and submits their updates back to the central repository.

Note that the author considers CM tools with proxies as one central repository. Though the proxies could be considered distributed repositories, their primary purpose is to increase performance and they are generally assumed to be connected to the central repository at all times.

b. Advantages

Because all access control and user authorization mechanisms are in one location, they are easier to set up correctly, manage and audit. One central system is easier to harden and physically protect than many distributed systems. Furthermore, the state of the system and the tool can be determined directly at any point in time.

c. Disadvantages

One repository means one single point of failure, though using a comprehensive backup process mitigates this risk.

2. Peer-to-Peer or Hierarchical Distributed Repositories

a. Description

Having one central repository is not ideally suited for teams that are large, geographically dispersed and frequently disconnected (such as open source development efforts) for a number of reasons. Firstly, the small, distributed teams often work on small subparts of the overall projects and only need access to a subset of the repository. Secondly, centrally administering user authentication and access privileges for users with whom the administrator is not familiar is difficult. Thirdly, if there is only one repository and users are not able to connect to it, users' workspaces quickly become out of date. Being able to create a distributed partial replica addresses most of these concerns, which is the reason why this architecture was created.

b. Advantages

Multiple repositories provide some redundancy; if one repository were to be corrupted or compromised, recovery would be more likely. Also, pushing the administration of users to the local administer who is more likely to be familiar with the users and their needs increases the likelihood that the authorization and access control will be managed appropriately.

c. Disadvantages

Distributed repositories are only as strong as their weakest link. If malicious users are able to compromise one of the replicas because of poor administration or by setting up an imposter replica, that replica can easily compromise other replicas when other replicas incorporate its changes. One of the tools that uses this replica architecture (OpenCM) has controls that enable users to identify a compromise and its source, but the tool doesn't actually prevent the compromise from occurring. Unless a strong audit procedure is in place, the compromise is likely to go unnoticed.

B. REPOSITORY STRUCTURE

Files, revisions, and metadata under CM must be stored. The format chosen will impact Goal 7 ("Protect integrity of items in repository"), Goal 1 ("Control all evidence and tools"), and Goal 3 ("Maintain consistency of evidence mapping").

1. Use Operating System's File System

a. Description

Each file or revision under CM is stored as a file in the underlying operating system and can be browsed through the operating system's file system browser (subject to access constraints). Some tools store revisions using the RCS format, which stores the most current version in full and stores only the differences of previous versions. Some tools supplement the RCS format with additional meta-data may be stored in a separate file or database.

b. Advantages

Using the operating system's file system makes the organization of the files completely transparent. This transparency reduced a malicious user's ability to use a backdoor in the CM tool to hide or change certain files or types of files without detection. Also, the operating system mechanisms for safeguarding files (e.g., access control lists, checksum tools) can be used; though the OS mechanisms are not fail-proof, they have been more thoroughly tested than a CM's tool proprietary mechanisms. Furthermore, corruption of the meta-data information does not corrupt the files themselves. The organization of the files still would provide basic information about the configuration and versions.

c. Disadvantages

Any compromise of the host system (e.g., a buffer overflow that disables access control mechanisms) provides immediate, transparent access to the CM files. Because the file system is in such widespread use, the number of known attacks against it is continually growing. Using the operating system's file system also limits the granularity of the access control mechanism to that of the operating system (i.e. read, write, execute), when many situations may want finer grain controls (e.g., "write access only if file is not frozen"). Most seriously, careless or malicious users with administrative

privileges can open and change the files under CM directly from the file system browser (or add a file to a folder in the CM hierarchy), rendering the CM tool's state information inaccurate in a subtle, perhaps indiscernible way.

Also, RCS has a number of vulnerabilities, including the lack of checksums.

2. Other File System or Database

a. Description

Each file under CM is stored in a proprietary file system of the CM tool that is hidden from the operating system. To understand the file structure, the files need to be viewed through the CM tool's browser.

b. Advantages

Because the files are stored in a proprietary file system, their immediate structure and contents is not available to someone who gained access (legitimately or illegitimately) to the CM tool's data file. This provides some deterrence against valid users changing the files directly through the file system, since they must go through the tool to make any meaningful changes to the files. Also, because the tool manages the file system, the tool can provide finer grain access control than what the operating system provides.

c. Disadvantages

Even if the CM tool uses a proprietary file system or database, the data is ultimately stored as a file in the operating system's file system and thus is reliant on the operating system's enforcement of access control. Thus this structure still has some of the disadvantages listed above for: "1. Use operating system's file system." In addition, motivated users are unlikely to be deterred by the complexity of the data storage system; they would spend the time and energy required to understand how the data is stored and how to manipulate it meaningfully.

Another drawback of using a proprietary file system or database is that the availability of robust recovery tools is typically smaller for an application specific file structure than for common, widely used operating systems.

3. COTS Database

a. Description

Some CM tools store both their files and their metadata in commercial, off-the-shelf database software products such as Oracle, Sybase, and SQL Server.

b. Advantages

The advantages that come with a powerful database product include querying and reporting capability that can be used for audit purposes, atomic transactions (covered as a separate feature area below), and finer-grained access control than provided by a file system.

c. Disadvantages

Since databases can be accessed via multiple means (e.g., web scripts, direct SQL, remote connections), using a database includes many of the disadvantages of using the operating system's file system: malicious or lazy users can circumvent the CM tool and access the data in the database directly, making changes that aren't logged and tracked appropriately. Also, as we found with using the operating system's file system, a vulnerability discovered in the database software product may leave the entire CM data at risk.

C. USER AUTHENTICATION

User authentication involves setting up users and groups and then authenticating them when they attempt to connect to the CM tool. Solid user authentication is relied upon for almost everything, but especially for Goal 4 ("Ensure that only authorized changes are made").

1. Use Underlying Operating System's User Authentication

a. Description

Many tools leave user authentication to the operating system. If a user is logged in, the tool considers them authenticated and merely uses their user login as their username for the CM tool.

b. Advantages

Just as using the operating system's file system to store individual files increases transparency and decreases the likelihood of a CM-tool based backdoor, so does using the operating system's user authentication mechanism. Most operating systems allow administrators to enforce principles such as good passwords (e.g., requiring passwords to be at least a certain length, requiring users to change passwords on a regular basis) and least privilege (e.g., granting users the least amount of privilege that is required by their role). Some operating systems also allow augmentation of user passwords with other authentication methods that rely on something the user has (e.g., a smart card) and something the user is (e.g., biometrics).

c. Disadvantages

But the strength of the user authentication is only as strong as the features the operating system provides and as the administration. An administrator who doesn't actively tighten down the authentication mechanisms leaves the system and the CM tool with weak user authentication. We would hope that a CM tool being used by a team building a high assurance product would be pro-active in maximizing the power of the operating system's authentication mechanism.

One limitation of using the operating system's own mechanism is that the tool is limited to managing the kinds of objects that the file system knows about—namely files and directories. The tool would not be able to restrict a user's access to part of a file, for example.

Another drawback is that giving users accounts on the system itself may provide them with additional ways to gain illegitimate access to other resources on the system besides the CM tool.

2. User CM Tool's Own Authentication Mechanism

a. Description

Some CM tools have built their own user authentication mechanism. The tool itself gives users names and logins. Software administrators control their access to the tool.

b. Advantages

When the CM tool manages user authentication, users do not need to have general access to the system on which the tool is running. This limits their privilege to the tool itself (baring any vulnerabilities in the tool), and thus reduces the damage they can do to the tool and its data.

c. Disadvantages

Implementing a rich authentication mechanism is not a trivial task. Given that most CM tools were built on the assumption that team members are trustworthy, the range of functionality provided by internal authentication is quite weak. Strong password-based authentication requires features such as password length minimums, password complexity, restrictions on repeated guessing of passwords, ability to check the strength of passwords, logging of all log-in attempts, prevention of log modifications, and protection of stored password hashes. Most CM tools choosing to implement their own authentication do not include all of these features. Furthermore, because the CM tool itself and its data must rely on the underlying operating system's access control mechanism at some level, CM tools end up with many of the disadvantages described above.

3. Use Public Key Encryption, Managed by CM Tool

a. Description

One method of implementing user authentication is to use public key encryption. For a good introduction to public key encryption, digital signatures, and certificates, see Netscape's "Introduction to Public-Key Cryptography" [PKCR98].

b. Advantages

This is a special case of the tool managing user authentication, so it has all the advantages described above for using the CM tool's own authentication mechanism (Section IV.C.2). Using public key encryption may also provide a mechanism that is stronger than the operating system's authentication.

c. Disadvantages

All the issues around public key encryption are issues here as well, including safeguarding of private keys, maintaining the integrity of public keys, and managing the revocation of compromised keys.

D. ACCESS CONTROL GRANULARITY

Access control granularity describes the ways in which an administrator can grant or limit a user's or group's access to objects in the CM tool. The granularity must be at a level that allows the tool to be used to fulfill Goals 4 ("Only Authorized Changes") and 2 ("Separation of privilege with change and commit").

1. Definable at the Repository Level

a. Description

Users are either granted or not granted access to a specific repository.

b. Advantages

None.

c. Disadvantages

In order to ensure that only authorized changes are made and that the user that makes the change is not the one that commits the change, administrators need to be able to grant more granular access control.

2. Definable at the Branch Level

a. Description

Users are either granted or not granted access to each branch in a repository. To support ensuring that only authorized changes are made and to support separation of privilege, branches can be set up to represent different hierarchical promotion levels (e.g., development, testing, verifiedByPerson1, verifiedByPerson2, and released). Access to each branch can be restricted to only the users who are intended to approve promotion for that stage of the process.

b. Advantages

Using branches is easy to implement and audit.

c. Disadvantages

But branch-level access control may not allow for enough granularity of control at the lower levels to help ensure that the evidence is created correctly.

3. Complex Access Control Based On Configuration State and User Roles

a. Description

Tools that include significant lifecycle functionality and have a notion of user roles allow access to be defined at a state and/or role level. For example, a document may be locked for writing until a user in the Approval Role "approves" it and moves it to a new state. Then users in the Development Role may write to it as long as it is in the QA state. These access rights can also be limited to a specific design part of the system being built.

b. Advantages

Teams are most likely to be able to enforce a "least privilege" policy using tools that have complex access control based on configuration state. The state and relationship rules most easily represent the types of restrictions that high assurance teams want to put on their teams.

c. Disadvantages

Because the mechanism is more complex, the likelihood of administrator errors is increased.

E. STORAGE OF ACCESS CONTROL INFORMATION

Access control information must be stored in order for the tool to enforce it. The integrity of the access control information is critical for maintaining Goal 4 ("Only Authorized Changes"). If a malicious or even lazy user can change the access control information, there is no way the tool can achieve Goal 4.

1. Stored In File

a. Description

Some tools store the access control information unencrypted as a simple file which the tools checks before allowing access to a specific area of the tool.

b. Advantages

None.

c. Disadvantages

Anyone who is able to gain illegitimate access to the file is able to manipulate access controls without being detected. Administrators must be trusted to appropriately handle the access controls. If an administrator's authentication mechanism is compromised (e.g., password guessed), no assurance is provided.

2. Stored In Digitally Signed Object Structure

a. Description

The access controls are digital signed using the repository's private key. Any change to the access controls by someone other than an administrator in the repository can be detected because the signature won't match (i.e. specifically, the hash in the digital signature won't match the hash of the compromised access control information).

b. Advantages

Anyone who is able to gain illegitimate access to the access control files would not be able to change them without being detected.

c. Disadvantages

Administrators must be trusted to appropriately handle the access controls..

3. Stored Encrypted In Database

a. Description

Tools that use a COTS database to store items under CM often encrypt the access control information and store it in the database as well. The author was not able to determine what type of encryption such tools used, nor how they managed the keys.

b. Advantages

Because the information is encrypted, anyone who is able to gain illegitimate access to the access control files in the database would not be able to manipulate the information usefully.

c. Disadvantages

Administrators must be trusted to appropriately handle the access controls. If an administrator's authentication mechanism is compromised (e.g., password is guessed), no assurance is provided.

F. CONFIGURATION DEFINITION AND ENFORCEMENT

A configuration is a set of files logically belonging together. For example, the latest version of a security policy, the related security model, and the related proof are considered part of the same configuration. When the model and proof are updated, the new model and proof, plus the original security policy, are considered a new version of the same configuration.

Goal 3 ("Maintain the consistency of evidence mapping") depends on the tool's ability to identify the elements of a configuration and to maintain them as a coherent set. For high assurance development efforts, the requirements are quite significant. A configuration should include all of its appropriate upstream counterparts (i.e. all related documents that are typically created before the given document), and all members of a configuration should correspond to each other formally (e.g., the proof should be a proof of the model in the configuration; the model in the configuration should be a formalization of the policy in the same configuration). Ideally, the CM tool would help enforce these rules.

The code correspondence stage of formal high assurance efforts requires lines of code (i.e. parts of a file) to be linked to lines of the formal specification (i.e. parts of another file). None of the tools allow you to define relationships within the file contents.

1. Each File Has Its Own Separate Version History; Weak Support For Grouping Files

a. Description

Each file develops a version history of its own. In order to figure out which files were committed at the same time; one has to look for all files that were committed by the same user at the same date and time with the same comment. Tags can be assigned to label a set of files as well.

b. Advantages

None.

c. Disadvantages

Subversion of what users consider a logical configuration is fairly easy for a malicious or careless user. The user could manipulate date/time stamps and make a file appear to be part of a configuration to which it doesn't belong. Since tools with this implementation tend to have non-atomic transactions, a tagged configuration may not represent the complete configuration.. Furthermore, there is no support for ensuring that a configuration has the right type of files in it (as described above).

2. Set Of All Files at a Given Moment In Time

a. Description

These tools track files that are part of the same configuration using transactions. Files that are committed by a user together are bond together via a label or a transaction number. The tool maintains history on all changes made to these files. Many tools with this implementation are loosely integrated with a requirements/bug tracking tool. The integration usually provides a rudimentary way to provide more information about a set of changes (such as "in response to bug 123")

b. Advantages

Being able to identify changes to *sets* of files is important for achieving Goal 3. Often in high assurance development efforts, team members need to verify that a change in one file (e.g., change in specification) resulted only in a change to specific files (i.e. those related to the specification) and not to changes in other files (i.e. those not related to the specification). Unexpected changes in unrelated files might be evidence of a backdoor inserted by a malicious developer, so being able to track all the files changed at one time is very important.

Integration with requirements/bug tracking tools improves the information available about a change, making it more likely that an unauthorized change would be noticed.

c. Disadvantages

The primary shortcoming of this implementation method is that there is no way to specify different types of relationships between files in a set. Knowing that the files belong together is helpful, as described above, but as the set of files grows, the relationships within the set become more complex. For example, a code file might be related to only one file in the formal specification. These tools do not support relationships within a set of files.

3. Set Of All Files and Their Related Changed Documents and State History

a. Description

Tools that have full lifecycle support usually include a mechanism for defining and relating files in richer ways than simply grouping them together into sets. One way this is implemented is through default relationships, such as “in response to” and “affected by”, and user-defined relationships, such as “proves”. Users can establish a relationship between files or between a file and a change document (see Lifecycle support section).

b. Advantages

Being able to define precise relationships helps document a development effort, helping the development team to better understand complex sets of files and outsiders to more quickly learn the intricacies of a code base.

c. Disadvantages

The additional complexity of the CM tool itself to support relationships increases the likelihood of errors in the enforcement of such relationships. Also, establishing these types of relationships adds some complexity to configuration set up and maintenance, but if the relationships were to be maintained manually anyhow, then automating it is not a significant disadvantage.

G MAKING HISTORY IMMUTABLE

One key role of a configuration management software tool is to track history regarding a development project. A CM tool should be able to answer questions like, “What did this set of files look like on May 29?” and “What version of file X was used in

release 1.02?" History should be protected from modification or the answers to these questions become unreliable and the tool becomes unable to achieve Goals 3 ("Maintain Consistency Of Evidence Mapping"), 4 ("Only Authorized Changes"), and 8 ("Protect Integrity Of Items In Repository"). Tools provide different types of protection against information that should be immutable such as file contents, commit history, and user access information.

1. Limit Changes To Administrative Users

a. Description

Some tools restrict changes to immutable content to the highest level of administrator user, using the tool's user authentication mechanism to enforce this restriction (see "IV.C. User Authentication" above). These users can then change history in subtle ways that would not be detectable by team members. A team could set the policy that the administrator is not supposed to change any of the historical data, but the tool would not enforce this policy.

b. Advantages

None.

c. Disadvantages

Administrative users may not be trustworthy. Administrative users may delete or change historical data by mistake. Furthermore, if a malicious user compromises the access control mechanism, that user would be able to modify or delete the entire history.

2. Stored In an Append-Only Database

a. Description

Some tools store immutable content in a proprietary append-only database. Not even administrative users can change items (e.g., files, transactional history).

b. Advantages

The append-only database enforces a mandatory access control policy of "no rewrite" and provides more assurance than a discretionary access control policy that allows administrative users to change the data.

c. Disadvantages

The append-only database resides in the file system of the operating system on which the CM tool is running; thus it is susceptible to compromise as described above in IV.B., "Repository Structure." Furthermore, if the database were to become corrupted, the "append-only" nature would likely prevent an administrator from being able to fix the problem.

3. Enforced by Cryptographic Hashes and Digital Signatures

a. Description

Another implementation is to use public cryptographic techniques to verify that the immutable content has not been changed. First, the content is named by its cryptographic hash. Since the hash is unique (discounting the very rare chance of a collision), any change to the content will result in a mismatch between its name and its contents – a clear sign that the content has been compromised. The hash is protected from change with the digital signature of the server on which the contents was created. Note that the content itself is not actually protected from change by either the hash or the signature, but since the signature protects the hash, re-computing the hash and comparing it to the protected hash would detect any change to the original content.

b. Advantages

This mechanism will effectively detect changes to frozen content, and could be used in conjunction with one of the other protection implementation methods above to create a strong enforcement of frozen content.

c. Disadvantages

Because this implementation method does not actually protect the content from change, it has the same disadvantages of whatever method is used to do the protection. However, its ability to recognize a change reduces the seriousness of these disadvantages. In addition, because this method relies on public key encryption, it has the disadvantages listed above in IV.C.3, "Use Public Key Encryption."

H. CHANGE TRANSACTION ATOMICITY

A typical operation of a user of a CM tool is to commit a "change" to the repository. A "change" to a user consists of a set of changed files that fulfill the requirements of a change request. Being able to identify all the components affected by a change is Goal 6.

To the CM tool, committing the change means copying the files from the user's workspace to the appropriate place in the repository hierarchy. Occasionally, while the software tool is performing the commit, some sort of error occurs that prevents the tool from completing the commit (e.g., the network between the user's machine and the server is broken, CM tool tries to write to a bad sector). Different CM tools respond to this situation differently.

1. Not-Atomic

a. Description

If the tool does not support atomic transactions, then some of the files may be committed and others may not. The reason the tools do this is historic. Older CM tools considered the file to be the base unit for CM (see IV.F.1, "Each file has its own separate version history; weak support for grouping files"). When customers wanted to be able to commit multiple files, the tools added the ability to do so by doing a series of sequential commit of each file.

b. Advantages

There are no advantages from the perspective of a high assurance team.

c. Disadvantages

Clearly, committing some subset of files of a change can leave the repository in a state that violates the consistency of the evidence mapping (Goal 3) and/or represents an unauthorized change (Goal 4). Malicious users can craft an error that causes an incomplete commit specifically to manipulate the state of the repository.

2. Atomic

a. Description

If the tool supports change transactions as completely atomic transactions, no files would be committed and the user would have to redo the commit.

b. Advantages

Guaranteeing atomicity ensures that anyone looking at a version of the repository, sees a consistent set of data, and supports Goals 3 and 4.

c. Disadvantages

None.

I. LIFECYCLE SUPPORT

CM tools provide varying degrees of lifecycle support and even define the scope of the term lifecycle differently. Some tools providing lifecycle support do support the full lifecycle of activities of a development project (e.g., Requirements Development, Design, Product Development, Test, and Release). Other tools making the same claims mean only that they have mechanisms to support the sequence of states through which a file (or set of files) of programming code passes through between creation and release (e.g., development, testing, release 1, bug fixing, release 1.).

High assurance development projects need to exercise control over the entire lifecycle of activities (i.e. the first interpretation), not just the activities between development of code and release (i.e. the second interpretation above).

Clearly, there are manual ways outside of CM tools to control the lifecycle; in this section we focus on the ability of CM tools to support lifecycle control.

1. Using Branch Hierarchy; May Include Links to a Requirements Tracking Tool

a. Description

The most basic way to support lifecycles in CM tools is to set up the branching hierarchy to represent the lifecycle states through which files or configurations must pass. Each state becomes a branch. For example, if the security model goes through states: “Draft,” “Reviewed Draft,” “Verified By Person 1,” “Verified By Person 2,” and

"Final," then the "Final" branch is the core branch and off of the "Final" branch you create a "Verified by Person 2" branch, off of which you create a "Verified by Person 1" branch and so on. You implement the concept of "approval" by limiting the ability to promote a file from one branch to the next to the person with approval authority.

Different document types in a high assurance development project (e.g., security policy, security model, model proof, code) may have different lifecycles. But if you want the CM tool to enforce the links between these different document types (i.e. maintain them in the same configuration), then the lifecycles of each must have the same end states (i.e. the branches representing each must merge at some level before the last stage). For example, the security model and the code could merge at the "Verified by Person 1" branch.

Many of these tools claim to be integrated with requirements or bug tracking tools. In reality, the tracking tools and the CM tool tend to have superficial links, created after both tools were designed and implemented. The lack of tight integration means that the tools tend to have completely distinct user authorization models and access control abilities.

b. Advantages

All CM tools provide some branching mechanism; hence all tools provide at least rudimentary lifecycle support. Implementing the lifecycle in the tool (opposed to manually) means that the entire history of when files were promoted to what stage by whom is captured. Furthermore, using the tool to enforce the lifecycle means that the same process will be used for all files, increasing repeatability and ability for audit.

The link to the requirements tracking tool can be useful as long as its shortcomings in enforcing any type of assurance are recognized.

c. Disadvantages

In most tools, branching was created to support situations where part of the code needed to evolve in a different way than the mainline of code, for example to support a separate platform or a bug fix. Using branches to represent stages in a

development lifecycle is quite different, and may be very awkward to implement with some tools.

The real lifecycle for high assurance projects consists of more than sequential states where documents can be promoted with approval from the right person. For example, there may be a set of questions that must be answered and recorded before a document may be promoted to the "Verified by Person 1" stage. Since the implementation method using branches cannot represent this, the full lifecycle cannot be fully represented simply by using branches. As a result, lifecycle enforcement would be limited to manual checks to ensure that processes are being followed and repeatability would be based on the quality of the manual procedures and the discipline of the people following the procedures.

2. Lifecycle Stages With Associated Change Documents and Rules

a. Description

Tools that claim to support the full system lifecycle provide significantly more functionality. One tool describes its lifecycle support as follows:

> A lifecycle or workflow is a series of activities done in a specific order to enact change in your system. A lifecycle is made up of "states," blocks of activities that describe a major area of work. A typical development lifecycle may consist of the following states: Requirements Development, Design, Product Development, Test and Release. Each state contains entry and exit criteria, i.e., reasons for entering and exiting a state. Typically, a state cannot be considered finished until someone has approved the work activities, indicating that the work was accurately and effectively completed [PVCS03, *Technical Brief*].

You can use the exit and entry criteria, as well as the rules to represent complex requirements that you maintain in your real lifecycle process. For example, in order to enter a "Development" state, you can require the user to specify the Requirements Document that s/he is responding to. The Requirements Document and the code that the user creates are then linked together. Before the user's code can exit the development state and move to test, you can require the developer to fill out a "Changes Made" form, the manager to fill out an approval form, and the tester to link the changes

to the appropriate test files to be run against it. Once completed, these forms become linked together with the requirements and the code for future review and audit.

b. Advantages

This implementation method allows you to model more of your lifecycle process in the tool and thus provides more standardization, repeatability, and auditability.

c. Disadvantages

Setting up the complex lifecycle process appropriately, because of its complexity, may be difficult. And there is no way to confirm that the process is correct. Also, the CM tool code that implements such rich lifecycle support is complex. Increased complexity increases the likelihood of errors in the enforcement of the lifecycle.

J. EXPORT/IMPORT

Because the author believes that high assurance projects require both "Everyday CM" and "Trusted CM" in their CM Plan, the CM tool used for "Everyday CM" must have sufficient exporting and importing capabilities to achieve Goal 6 ("Implement the CM Plan"). Specifically, tools must be able to export the proposed baseline and changes to the "Trusted CM" team in an acceptable format and will need to able to import files from the "Trusted CM" team for modification of an accepted baseline.

The conditions under which the export/import functionality would be used in a high assurance project are an area for further research. Thus the detail below focuses on the export of the files themselves (not metadata), and should be considered merely as a starting point.

1. Straight Copy From File System or Database

a. Description

For tools that store CM files in the operating system's file system or in a standard database management system, exporting and importing would use the file system's move function or the database system's export function.

b. Advantages

These functions have been used extensively for years by a very large user base; basic errors are unlikely to still be unknown.

c. Disadvantages

The functions could be replaced with subverted ones, but by checking checksums with known clean versions, the threat can be minimized.

2. Import/Export Function In Tool

a. Description

Tools that have a proprietary way of storing the files under CM must also have a proprietary way of translating between that format and a more standard format, such as directories and files in the operating system's file system.

b. Advantages

None.

c. Disadvantages

Because the functions are not as widely used, they are more likely to have unidentified bugs. Checking for subversion is difficult or impossible.

K. THREADED DISCUSSIONS

Threaded discussions are a familiar fixture on the Internet. One of the tools evaluated provides a mechanism for threaded discussions (StarTeam). The definition of a threaded discussion in StarTeam's user manual [STWB03] is:

> A series of responses to a posted topic. Each conversation forms a topic tree with the topic as its root. It is called a threaded conversation because the tree hierarchy indicates whether a response is a reply to the topic or another response to that topic. By reading each response in a thread, one after the other, you can see how the discussion has evolved.

Threaded discussion can help a high assurance effort by providing another source of documentation about changes to the system. Since part of the CM Plan is to show "that the acceptance procedures provide adequate and appropriate review of changes to all configuration items," threaded discussions can help Goal 8 ("Implement CM Plan").

1. No Threaded Discussions

a. Description

No functionality for integrated threaded discussions

b. Advantages

None.

c. Disadvantages

Tool-based documentation of a change is limited to the user's commit comment, or, in tools that support lifecycle management, the contents of an electronic customizable "Change Form." These are important sources, but threaded discussions provide additional valuable information.

2. Threaded Discussions

a. Description

Some functionality for integrated threaded discussions

b. Advantages

Threaded discussions capture first-hand, time-stamped written discussions between team members about changes. Just as emails provide a trail of evidence that phone conversations do not, threaded discussions provide a level of detail that short comments and responses to standardized fields do not. Discussions are no substitute for either the comments or the forms, but they are an excellent addition to those two more formal sources. One additional benefit of threaded discussions is that someone who is familiar with the team members and the issues is likely to be able to detect a discussion that has been completely fabricated.

c. Disadvantages

None.

L. INTEGRITY VERIFICATION

In a high assurance project, the CM tool should help protect the integrity of the items in the repository (Goal 7) against both random errors (e.g., hardware failures) and malicious errors.

1. No Integrity Verification

a. Description

Most tools provide no specific integrity mechanism.

b. Advantages

There are no advantages to not providing an additional source of integrity and authenticity verification.

c. Disadvantages

Less assurance.

2. Integrity Verification Using Hashes

a. Description

One of the tools (BitKeeper) stores checksums for each file and revision.

b. Advantages

BitKeeper summarizes the advantages of this method in [BKWB03]:

> BitKeeper checksums each revision file and each delta of each revision file. […] So far, these checks have found multiple bad memory DIMMs, many NFS corruptions, Linux/XFS corruptions, and a few SPARC/Linux cache aliasing bugs. All of those errors are likely to go undetected in an RCS-based system such as Perforce, CVS, etc. RCS has no built-in integrity checks and is made worse by a file format that prevents the detection of the bad data until the system attempts to retrieve a version of the file containing the corrupted section of the version control file.

c. Disadvantages

Because the hash itself is not protected, this style of integrity verification will not detect changes made by a malicious user. A malicious user could change the hash to match the changed file.

3. Integrity Verification Using Protected Hashes

a. Description

One of the tools (OpenCM) provides an integrity check for items under CM by computing the cryptographic hashes of each item and protecting the hash with the digital signature of the person submitting the object.

b. Advantages

Using hashes does not prevent integrity loss, but it does enable users to verify that the integrity of an item has not been compromised. Clearly, being able to verify the integrity of items under CM is valuable to CM teams.

c. Disadvantages

There are no disadvantages from a high assurance perspective in having a way to verify the integrity of an item.

M. OTHER CM FEATURES TO CONSIDER

The evaluation results focus on feature areas that are the most relevant to high assurance projects looking for a CM tool. However, these are by no means the only important feature areas. Other areas are critically important to the success of all software teams' configuration management processes, including (partially from [DART00]):

- Stability
- Performance
- Available support
- Amount of administration required for set up and maintenance
- Amount of computer power required for set up and maintenance
- Cost
- Vendor reliability and viability
- Platforms supported
- Fault tolerance
- Scalability (if applicable)
- Customizability
- Usability

Any project evaluating CM tools would be wise to evaluate each tool in each of these areas.

V. CM TOOL EVALUATION

A. CM TOOL DESCRIPTIONS

A fuller description of each tools evaluated is included in Appendix A.

1. AccuRev

AccuRev is a newer CM tool that prides itself on its architecture that supports very flexible branching (called "streams") and enforces atomic transactions. It has minimal process management support.

2. BitKeeper

BitKeeper's most notable feature is its distributed, peer-to-peer replica architecture. Linus Torvalds, Linux's leader, started using BitKeeper for Linux in 2002 [BKLX03]. BitKeeper is free to open source efforts, but such efforts must provide BitKeeper with its metadata within 21 days of its creation and must respond to a request to make its repositories available to the public with 15 days of BitKeeper's request.[BKWB03, "Free Use License"].

3. ClearCase

ClearCase is the market leader and provides change management functionality in addition to the standard version control. ClearCase is part of a suite of products from Rational that implement Rational's "best practices" software development methodology, Unified Change Management.

4. CVS

CVS, a free, open source tool, is perhaps the best-known CM tool. Even though it has some serious shortcomings, CVS is used for dozens of open source projects, including Apache WWW server, FreeBSD, NetBSD, OpenBSD, GNOME, and Xemacs [CVSW03]. Subversion is the open source effort intended to replace CVS [CVSS03].

5. OpenCM

OpenCM, another free, open source tool, was created to fulfill the requirements of the John Hopkins team creating EROS, a high assurance operating system. Their unique requirements included the ability to support a geographically distributed and often off-

line development team, as well as the ability to provide more assurance than existing CM tools. The EROS team wanted more assurance in the areas of provenance tracking and integrity checking across potentially hostile replicates [OPAI02]. To provide the additional assurance, OpenCM employs cryptographic hashes for object naming and public key encryption for user and server authentication.

6. Perforce

Perforce is a popular tool in the academic community, perhaps because the company provides the tool for free to open source efforts (efforts that provide unrestricted read-only access and release their software under one of three GNU licenses) [PFWB03, "Open Source Contract"]. Perforce is known for its simple architecture and unique branching model that promotes outwards instead of inwards towards the tree's trunk.

7. Merant Dimensions

Merant now sells Dimensions, which was originally marketed as "PCMS Dimensions" by a company named SQL Software. Merant Dimensions (previously "PVCS Dimensions") is one of the CM tools with a rich set of process management features. Note that PVCS Professional, also from Merant, has a completely different source and history; it is not just a scaled down version of Dimensions. Many government agencies and government contractors use PVCS products. The US Navy has an enterprise license for PVCS products.

8. StarTeam

StarTeam is another high end CM tool. StarTeam has integrated threaded discussions and provides the option to encrypt data sent between client and server.

B. TOOLS BY FEATURE AREA AND IMPLEMENTATION METHOD

Table 5 below shows how the tools implement each feature area, according to the author's sources at the time she performed the research (see List of References). Each implementation has a risk rating based on the advantages and disadvantages summarized in Chapter IV. The risk rating is a quantitative estimate of the amount of opportunity (1=Minimal, 2=Some, 3=Significant, 4=Serious) the given implementation creates or

allows for exploits and errors that would negatively affect the likelihood of a team being able to successfully create the necessary evidence. A tool's total risk rating is simply the sum of the ratings from its component feature areas.

Remember that these ratings do not take into account the protected environment in which the tools should be employed, as described in II.D. above. High ratings reflect a lack of inherent risk mitigation, and represent the areas most critically in need of protection by mechanisms outside of the CM tool.

Feature Area	**Implementations**	Risk Rating	AccuRev	BitKeeper	ClearCase	CVS	OpenCM	Perforce	Dimensions	StarTeam
Total Risk Rating	*Total Risk Rating Minimum/ Maximum*	11/ 37	24	25	25	31	19	23	24	23
A. Repository Architecture	One central repository plus user workspaces	1	1		1	1		1	1	1
	Peer-to-peer or hierarchical distributed repositories	2[4]		2			2			
B. Repository Structure	Use operating system's file system	2				2		2		
	Other file system or database	3	3	3	3		3		3	
	COTS database	2								2
C. User Authen-tication	Use underlying operating system's user authentication	2		2		1		1	1	
	User CM tool's own authentication mechanism	3	3		3					3
	Use public key encryption, managed by CM tool	1					1			
D. Access Control Granularity	Definable at the Repository Level	4				4				
	Definable at the Branch Level	2	2	2			2	2		
	Complex access control based on configuration state and user roles	1			1				1	1

[4] The rating is for the peer-to-peer or hierarchical distributed architecture. If the tools are used with only a central repository, the risk rating would be the same as that of the central repository.

Feature Area	Implementations	Risk Rating	AccuRev	BitKeeper	ClearCase	CVS	OpenCM	Perforce	Dimensions	StarTeam
E. Storage Of Access Control Information	Stored in file	3	3			3		3		
	Stored In Digitally Signed Object Structure	1					1			
	Stored Encrypted In Database	2		2	2				2	2
F. Config-uration Definition And Enforcement	Each file has its own separate version history; weak support for grouping files	4			4	4				
	Set of all files at a given moment in time	2	2	2			2	2		
	Set of all files and their related changed documents and state history	1							1	1
G. Making History Immutable	Limit Changes To Administrative Users	4		4	4	4		4	4	4
	Stored In An Append-Only Database	2	2							
	Enforced By Cryptographic Hashes And Digital Signatures	1					1			
H. Change Transaction Atomicity	Not-Atomic	4				4			4	4
	Atomic	0	0	0	0		0	0		
I. Lifecycle Support	Using branch hierarchy	3	3	3		3	3	3		
	Lifecycle stages with associated change documents and rules	1		1	1				1	1
J. Export/ Import	Straight copy from file system or database	1	1	1		1		1		
	Import/Export function in tool	2			2		2		2	2
K. Threaded Discussions	No threaded discussions	2	2	2	2	2	2	2	2	
	Threaded discussions	0								0
L. Integrity Verification	No integrity verification	3	3		3	3		3	3	3
	Integrity verification using hashes	1		1						
	Integrity verification using protected hashes	0					0			

Table 5. CM Tools Evaluation Summary

VI. CONCLUSIONS AND RECOMMENDATIONS

A. DISCUSSION OF EVALUATION RESULTS

The evaluation demonstrates that most tools have comparable risk ratings (23-25) if deployed in an unprotected environment, with the exception of OpenCM, which presented the least risk at 19, and CVS, which presented the most at 31.

OpenCM minimized risk most effectively through its use of public key encryption in the areas of user authentication, access control, immutable content, and integrity verification. These four areas are especially important when a CM tool is used in an unprotected environment, making OpenCM clearly the best choice for efforts that want to provide the most assurance possible in such environments.

However, a high assurance effort introduces unnecessary risk by choosing a tool that is just "better than the others." Until OpenCM runs on a high assurance operating system, a high assurance effort using OpenCM must provide additional environmental protections such as physical security, limited user access, and separation from other development efforts. Ironically, these additional protections mitigate the other tools' disadvantages in the four key areas where OpenCM shines, making the others tools more comparable to OpenCM. Providing multiple layers of assurance is still valuable, however, so teams using OpenCM would still be introducing the least risk.

Though all the other tools, except CVS, have virtually identical total risk ratings, the source of risk is different for each tool. Furthermore, most tools have a unique, risk-minimizing implementation in at least one area. For example, StarTeam has threaded discussions, AccuRev has an append-only database to protect immutable information, BitKeeper provides integrity verification, and Dimensions and ClearCase provide superior lifecycle support.

B. RECOMMENDATIONS

The ideal tool for a high assurance project would have all the risk-minimizing implementations for a total risk rating of 11. There is no reason why such a tool could not exist, for none of the "best" implementations are incompatible with another "best"

implementation. Alas, no such CM tool currently exists. The best approach for selecting a CM tool for a high assurance development project is to:

1. Identify several tools that meet your project's requirements, including the requirements that are not specifically assurance-related.
2. Determine the implementation methods of each tool and identify the risks created by the disadvantages of these methods described in section IV.
3. Figure out the environmental requirements for each tool that minimizes the tool's risks identified in step 2.
4. Select one of the tools whose environment your project can support. Be realistic about what your team and situation will support; for example, will all work really be done on a private network at one site? Make sure that your team is prepared to support the required protections, or your effort will likely fail.
5. If none of the tools are sufficient, repeat the steps 1-4 until you find an adequate tool.

If the protection environment your team creates that supports your daily development requirements is deemed insufficient to protect your effort from subversion by a malicious insider, then you should create separate "Everyday CM" and "Trusted CM" systems and processes as described in Section II.D, using the tool selection approach above with somewhat different project and environment requirements.

Following these recommendations and an appropriate CM plan should enable an high assurance effort to exceed the Common Criteria's configuration management requirements for EAL7.

APPENDIX

A. DETAILED CM TOOL INFORMATION

This appendix includes detailed information on each of the tools evaluated in the thesis. The information included here is not meant to be comprehensive, but merely to allow readers to get a sense of what differentiates each product from a high assurance perspective. Performance is not mentioned, for example, because it does not have a special relevance to high assurance projects. Note that the depth of the information on each tool varies significantly based on how unique each product is and on the author's source and time limitations. The descriptions here usually do not include information that is provided in Table 5 ("CM Tools Evaluation Summary").

1. AccuRev

Information on AccuRev came from product literature on the company's website [ACRV03], an Internet demonstration of the product by AccuRev sales personnel [ARDC03], and follow-up phone discussions with the same AccuRev personnel [ARBM03].

AccuRev is one of the newer tools, created by a company that was founded in 1998 by people that worked on the ClearCase product. AccuRev is a middle-tier CM product (i.e. supports range of version-control functions with minimal lifecycle support). Below are some of the key AccuRev features.

a. Complete, Time-Safe Versioning

When a change is committed in AccuRev, all information about the state of the project under CM is recorded, including file contents and metadata, directory contents and metadata, workspace contents, project contents and structure. AccuRev makes this information immutable by storing it a proprietary, append-only database. No one can change history, not even administrators, making the information "time-safe" or safe from change over time. This is a great feature for high assurance efforts.

b. Completely Atomic Transactions

AccuRev designed commits of multiple files to be atomic, unlike other tools, which implement multiple-file commits as multiple individual file commits. This is another important feature.

c. Dynamic Streams

Instead of branches, AccuRev uses streams. Streams are stages through which files progress. Streams are "dynamic" because they can be changed as necessary during a project. For example, if the team decides that they need two QA streams halfway through the project, they can insert a new stream (e.g., "QA Stage 2") in the development process. They can later remove it. To do this with most tools is impossible. Though AccuRev markets this feature as a real strength because of the dynamic nature of most development team's methodology, it does not appear to help high assurance efforts, which (should) have a fixed, formal methodology. Malicious users could exploit the stream flexibility by, for example, removing an approval stage in a stream temporarily in order to avoid having to pass through it. AccuRev does allow you to limit the design of the streams to specific users, so this threat can be minimized.

AccuRev has plans in future versions of its product to allow users to include components in a configuration. For example, you could link a driver with a specific project. Implementing this with branches would be difficult.

d. "Integrated" Issue Tracking System

AccuRev has a companion issue tracking system that provides a rich set of functionality and that is easily customized. One can link a commit in the CM system to an item in the Issue Tracking System (for example, link a change in a several code files to a change request). Unfortunately for high assurance projects, the Issue Tracking does not has no security at all currently, leaving any user able to change information (e.g., change the contents of the change request). The next release *may* honor repository-level security.

e. Access Control and User Authentication

Access Control Lists can be defined at the stream level. There is no history kept on access control lists, so there is no way to see who had access to what at what time, though obviously any change made by a user would be recorded and auditable. User authentication information is stored in a server-side script.

f. Usability

AccuRev claims that its tool requires fewer people to administer than other tools do. This claim makes sense given the clean and simple user interface, including its visual representation of a project's streams. AccuRev provides a pleasant user experience for common actions like filtering changes, tracing a set of changes, committing files, and merging.

g. Cost

AccuRev's cost is about $1000/user, which includes first year support. Second year maintenance per user is $350.

h. Product Specifications

According to AccuRev's website as of May, 2003, the product has the following specifications:

- **Platform Support**: Alpha systems: Compaq Tru64 Unix (version 4.0 +); HP systems: HP-UX (version 11.0 +); IBM systems: AIX (RS/6000) (version 4.3.2 +); Intel/x86-based systems: Windows XP, 2000, NT 4.0, Windows 95/98/Me (full client support, server for evaluation only), Linux (kernel versions 2.0.36 +, RedHat 5.0 +), FreeBSD (version 3.3 +); PowerPC systems: Linux (version 2.2.6-15 +); SGI systems: Irix (version 6.2 +); Sun systems: Solaris (version 2.5.1 +)
- **Development Tool Integration**: Supports a variety of IDEs, and other tools, including MS-SCC with MS Visual C++.

2. BitKeeper

Information on BitKeeper came from BitWise's website [BKWB03], comparisons by OpenCM [OPCN02], overviews of tools in a CM book [BERC03], and LinuxWorld [BKLX03].

a. Peer-To-Peer Repository Architecture

BitKeeper's most unique feature is its peer-to-peer repository architecture. Every workspace is considered a repository and a user can "push" or "pull" changes to/from other repositories. A typical use of the flexible architecture is as a hierarchy of repositories, with the root repository considered the closest to release. Each repository can be used as a staging or approval point.

Though BitKeeper's architecture is typically used to support teams that are geographically distributed and even disconnected, a high assurance team could set up BitKeeper on a closed environment and take advantage of its architecture to separate regular development from the "Everyday CM" for the trusted computing base and to separate both of these from the "Trusted CM."

The distributed architecture has more redundancy in it than tools that rely on a central repository; if one repository fails, much of the information is likely to be stored on other repositories.

Multiple repositories does introduce the question of how the repositories establish and maintain trust with one another—or more specifically, how they prevent malicious users from passing an imposter repository as a real one. BitKeeper fails to provide any mechanism to detect imposter repositories [OPEL03]. Thus if a high assurance effort were to use BitKeeper, they would have to provide mechanisms using physical security, trusted users, etc.

b. Tracking Changes With Changesets

Like AccuRev, BitKeeper tracks changes at the configuration level in a time-safe way. Thus changes to multiple files are considered one unit of work (a changeset), and all information about the entire tree structure is stored. Whereas some tools require users to explicitly "tag" a moment in time, BitKeeper automatically tags for every changeset

c. Data Integrity With Checksums

In order to detect integrity problems (e.g., those caused by disk errors), BitKeeper keeps track of each revision's checksum. Systems without such a mechanism

(such as those based on RCS) are unable to detect bad data until the system actually tries to retrieve a file that includes the bad data.

BitKeeper's checksums are not protected, thus they do not protect the integrity of the data from malicious users; such users would merely change the data *and* the checksums.

3. ClearCase

Information on ClearCase came from ClearCase product information on Ration's website [CLCS03] and an interview with a long-time user of ClearCase and other Rational products [CLJS03].

ClearCase is the current market leader, with over 30% of the market according to a 2000 study by IDC [IRCM02]. It is on the more complex end of the functionality spectrum because it supports lifecycle management. Some of its key features are described below.

a. Part of a Software Methodology Framework

ClearCase is just one the many products that the company Rational sells that supports the Rational Unified Process® (RUP), a comprehensive framework for delivering software development best practices. Out of the box, ClearCase comes setup to support RUP.

b. Lifecycle Support

ClearCase lifecycle can be customized to support the customer's workflow,

c. Integration With Issue Tracking Tool

Rational's companion tool is called ClearQuest. Using both products together enables you "to enforce common, consistent processes for submitting, assigning, resolving and verifying modifications" [CLCS03]. ClearQuest only runs on Windows and Microsoft IIS.

d. Product Specifications

According to the materials on their web site in May, 2003, ClearCase has the following specifications.

- **Client Requirements**: Minimum: 64 MB RAM, 35 MB Hard Disk Space
- **Server Requirements:** Minimum: 128MB RAM, 70MB Hard Disk Space
- **Supported Web Browsers**: Microsoft Internet Explorer, Netscape
- **Supported Web Servers**: Apache, Microsoft IIS, Netscape
- **Supported Environments**: Windows XP Pro, Windows 2000, Windows NT, Windows 95/98/ME (client only), Compaq Tru64 UNIX, Hewlett-Packard HP-UX, IBM AIX, Red Hat Linux Intel, SCO UnixWare, Siemens Reliant UNIX, Silicon Graphics IRIX, Sun Solaris SPARC, Solaris Intel, SuSE Linux Enterprise Server for IBM S/390 and zSeries
- **Product Integrations**: IBM VisualAge for Java, IBM WebSphere Server, IBM WebSphere Studio Application Developer, Borland Jbuilder, Microsoft Visual Studio, Visual Basic, Visual C++, Visual J++, Visual InterDev, Sun Forte for Java and C++, Sybase PowerBuilder, All SCC-compliant tools

4. CVS

The information gathered for CVS comes from CVS' website [CVSW03], the official CVS manual [CVSC03], a book on using CVS [CVSF99], interviews with users of CVS [CVSD03] and the website of Subversion, the open source project that is attempting to take over the CVS user base by providing a similar product without CVS' glaring omissions [CVSS03].

CVS is one of the most widely used CM tools, especially in the open source community, not least of all because it has a long history of being free. CVS was initially a collection of scripts, posted to the Usenet newsgroup comp.sources.unix in 1986, designed to improve the dominant version control tool at the time, Revision Control System (RCS). RCS provided a format for tracking changes to files. CVS added the ability to track files into a project, allowed parallel development, and (in the early 1990s) network awareness. Along the way, CVS was rewritten in C.

According to Fogel, CVS has become the "free software world's first choice for revision control" because there is a synergy between the way CVS encourages a project to run and the way free projects actually do run. As evidence of the synergy, Fogel points

out how convenient CVS makes providing read-only access to the world and generating patches to the frequently changing source. High assurance efforts need to carefully control both access to their repository and changes, suggesting less "synergy" between CVS and high assurance efforts.

CVS has several shortcomings in the areas of configurations, system access, and user authentication, discussed below.

a. Configurations

When a commit of multiple files is performed in CVS, the system knows that those revisions were performed together because they have the same date/time stamp, the same commit comment, and, if the user uses a tag, the same tag name. None of these provide any protection. Not only are date/time stamps notorious for being easily manipulated and commit comments obviously repeatable, but one can never be sure that all files in a set were committed because CVS doesn't support atomic transactions. Thus the tagged version that consists of fifteen files may have been a commit of twenty files that got interrupted by a network outage.

b. System Access and User Authentication

As CVS' own manual states, "Once a user has non-read-only access [and, in previous versions, read-only access] to the repository, she can execute programs on the server system through a variety of means. Thus, repository access implies fairly broad system access as well" [CVSC03, Section 2.9.3.3]. Also, passwords for clients are stored in a "trivial encoding" on the client side and transmitted in this encoding. In summary, as the manual puts it,

> Anyone who gets the password gets repository access (which may imply some measure of general system access as well). The password is available to anyone who can sniff network packets or read a protected (i.e., user read-only) file. If you want real security, get Kerberos.

5. OpenCM

The information on OpenCM was gathered from three papers written by the OpenCM creators [OPCN02, OPAI02, OPEL03], from email and phone discussions with

the creators [OPJS03, OPJL03, OPJV03], and from the OpenCM's User Guide [OPEN03]. More research on OpenCM is required to fully analyze its implementations and protocols; the descriptions here should be considered only a starting point.

OpenCM was created to fulfill the requirements of the John Hopkins team creating EROS, a high assurance operating system. Their unique requirements included the ability to support a geographically distributed and often off-line development team, and the ability to provide more assurance than existing CM tools. The EROS team wanted more assurance in the areas of provenance tracking and integrity checking across potentially hostile replicates. To provide the additional assurance, OpenCM employs cryptographic hashes for object naming and public key encryption for user and server authentication. Specific details below.

OpenCM is still relatively unproven, and some aspects of the features described below may not be fully implemented yet. As of June 2003, OpenCM was in alpha (version 0.1.2alpha5pl2-1). Though OpenCM is not as widely deployed as the other tools evaluated, OpenCM has been self-hosting for more than a year and OpenBSD is using OpenCM to maintain a duplicate repository [OPJS03]. Also, many of the major open source operating system efforts have asked whether OpenCM could be incorporated into their release.[OPEN03], showing that the tool has at least a potential market.

a. Repository

For a given development project, one server is considered the central repository and creates the main branch. Other servers can be set up as replicates of the central repository, or of existing replicas, though OpenCM can be setup to have exactly one repository. Each OpenCM server is responsible for authenticating its users.

b. Working Disconnected

The replica repositories do not need to be connected to the central repository at all times. They can periodically connect and get and send updates. When the central repository accepts the updates (more on this below), the entire history of commits from the replica (and perhaps from its replicas) is stored on the central repository, providing a complete history. Just as replicas can work disconnected from the central

repository, users can work disconnected from their local repository and make multiple local commits. These local commits are uploaded into their repository when they reconnect. Communication between replicas is done using TCP/IP and a specific OpenCM protocol.

c. Server Repository Naming

OpenCM Repositories or Servers are named by the hash of the server's initial public key. The public key is used because it will usually uniquely identify the server (though since each server generates its own public/private keys, there is no guarantee of uniqueness). Servers also generate public/private keys for their users. User public keys are stored on the server without encryption; private keys are protected by the key's user's password.

OpenCM Servers need to know how to contact other servers higher up in the replication chain. OpenCM plans to use DNS "repository registry" [OPAI02, p. 6], so server_registry.opencm.org will resolve to an IP address.

d. Object Naming

Frozen objects (i.e. content objects that never change such as a version of a file) are named by the cryptographic hash of their contents. Mutable objects are named with a URI of the form: opencm://server-name/swiss-number. The server-name is described above. The swiss-number is a cryptographically strong random number generated by SSL. The server is responsible for ensuring that there are no collisions between swiss-numbers on its machine; collisions with other servers are not a problem since the name of the object includes the server's name.

Only the originating repository can make changes to mutable objects. Every time a mutable object is changed, the server digitally signs the change. Part of the information signed includes the mutable object representing the authenticated user making the change.

e. Home Directories

Given that hashes are not very human-friendly, OpenCM uses a mechanism called home directories that allows users to map the hashes to names. "Each

user maintains total control over entries" in his/her home directory [OPEN03], but one can setup common directory namespaces that several members can modify. A malicious insider could manipulate the human readable names in a common namespace in an attempt to trick other team members into believing that one file is actually a different file. OpenCM has two features to help manage this risk. Firstly, an additional directory with more restricted access can be set up that maintains the correct mappings and serves as a check against the working directory. Secondly, history can be kept on all changes to names in home directories, providing an audit trail.

f. Users, Authentication and Access Control

OpenCM currently relies on SSL user authentication, but they are considering moving to SSH [OPNV03]. Only a user in the administrators group can create users. To create a user, the administrator must have the user's X.509 certificate in PEM format; the manual suggests emailing the certificate to the administrator. Users are requested to secure their private key with a passphrase. Users can remove their passwords (though they are strongly discouraged from doing this in the manual). Administrators can create groups in OpenCM that consist of a set of users. Group membership is transitive.

Access controls for users and groups are initially determined by the user that created the user or group. Access can be defined at the level of the mutable object (e.g., file, revision), since each object has a read and a write group. In addition, each repository has a higher-level access control list that determines whether the user/group has *any* access to the repository; this list overrides any object-specific rights the user/group may have.

Because each object has its own read and write groups, OpenCM can support the principle of separation of privilege by preventing the creation of an "all-powerful" administrator; instead, read and write privileges can be distributed among a set of administrators. The one power that all users in the admin group have is the ability to change a user's key; OpenCM plans to add auditing to this function.

g. Configurations

OpenCM distinguishes itself from CVS in one way by claiming that OpenCM has "real configurations"—i.e. configurations are sets of files, not individual files linked by a time/date stamp or a tag. In OpenCM, a configuration is a mutable object and is represented in part by an array of the names of the frozen content files. Every time the configuration is changed, the server signs the changed object.

h. Branches and Changes

OpenCM uniquely numbers branches. Users can add tags (i.e. names) to specific branch versions. These names are stored in a user's home directory. OpenCM recommends that you maintain development and audited branches for high assurance systems.

i. File System

OpenCM stores the mutable and frozen objects in a proprietary format. OpenCM offers several different options for storing objects, including both flat files (where one file is one object), and a delta storage strategy. The underlying operating system's access control lists protect the file store.

j. Recovering From Key Compromise

When it is discovered that a user's key has been compromised, the recover strategy consists of:

- Disabling the user's write access to the repository (using the overarching access feature described above),
- Auditing the repository to see what the user has done (which is easy to do since the user object is part of the information signed by the server every time a revision is created)
- Generating and installing a new key for the user

When a server's key is compromised, there is a window of opportunity for an attacker to set up an imposter server using the stolen key. But for the attacker to have any impact, the attacker must also be able to convince clients that its repository is legitimate by:

- Getting its IP address to the client and making the client believe that it is the real IP (or rerouting all traffic coming from the client destined for the real IP to its own IP)
- Being able to authenticate users by obtaining their keys

OpenCM provides several mechanisms to help minimize this window and this risk. Once a compromise is discovered, the server's key is put on the revocation list. The server then creates a new key for itself, and signs that key with its *offline* private key; the offline keys are distinct from the public/private keys that the server uses to sign changes. The offline key is used only to sign key updates. Once these changes have been made, the imposter will no longer be able to maintain its cover.

k. Guarantees

In [OPEL03], OpenCM's creators claim that OpenCM provides several guarantees. The guarantees are listed below, along with our best explanation for how OpenCM provides these guarantees.

Guarantee	**Explanation (of Author)**
1. The user can verify that any object obtained from a repository is valid. By "valid," we mean that an integrity check can be performed that reveals whether this object is complete, and that an authorized modifier of the branch checked it in. Valid does not imply correct – verifying the code is beyond the scope of OpenCM.	• Verify that object was checked-in by an authorized user: Decrypt digital signature using repository's public key. User object is part of the information signed, and can thus be verified. • Integrity check for completeness: Hash object and compare to hash in the decrypted signature. If the hashes match, the integrity is sound.
2. While all objects received can be authenticated, no guarantees are provided about whether the object is up to date unless the user obtains it from the originating repository. If the object is obtained from a replicate repository, it is guaranteed to have come from earlier valid state of the branch.	• OpenCM's naming and signing when users commit new or changed objects ensures their integrity and authentication. Since only objects that have been committed can be replicated to another repository, an object from a replicate repository is guaranteed to have come from an earlier valid state of the branch. See "Recovery From Key Compromise" above for an explanation of what happens if the server's key is compromised.

Guarantee	Explanation (of Author)
3. If a user's authentication key or client is compromised, total integrity exposure is limited to the set of branches that the user can modify; OpenCM as a whole is not compromised.	• OpenCM as a whole cannot be compromised by someone with a compromised key because that user would only have authority to make changes to branches to which the key's user had access.
4. Integrity verification is designed to be possible even if the user obtains certain types of partial copies of a branch. For example, the user may choose to replicate only selected versions of a branch, and can validate that the versions obtained are authentic.	• Partial versions of a branch can be validated using the digital signatures, as in Guarantee 1 above.
5. Provided the originating repository is not compromised, the complete history of each branch originating at that repository will be available from that repository. This has implications for merge management.	• All history will be available because all commits done by users in their individual workspaces are stored when the users commit to their local replica and all commit history is transferred when a local replica branch is merged to another replica's branch.
6. The repository records authentication information for every change. In the event of user key compromise, this information is sufficient to allow audit of suspicious changes.	• Since every change is signed by the user making the change, all changes made by a specific user can be identified and audited
7. Impersonating a repository requires both stealing the repository's private key and compromising the IP routing mechanisms near the client	• Stealing repository's private key: In order to make any changes on a repository, the server must have the private key. (An imposter repository that cannot make changes can only provide users with a valid, but perhaps not up-to-date, version.) • Compromising the IP routing: There is a registry that maps server replica names to IP addresses. This mapping would have to be compromised or the traffic coming in/out of a specific client would have to be captured and rerouted.

Table 6. OpenCM Guarantees Explained

6. Perforce

The author gathered information on Perforce from product literature on Perforce's website [PFWB03], and interviews (including a demonstration) with two users at an active Perforce installation [PFWG03].

Perforce is a popular commercial tool in academia, in part because open source projects (as most academic projects are) can license it for free. Perforce is one of the easier tools to understand quickly because of its branching model and its use of the operating system's file system to store files under CM. Some of its key features are described below.

a. Branching Model: "Inter-File Branching™"

Most CM tools promote towards the trunk: the user workspace is considered a leaf on the tree. As configurations are moved from user workspace to test to release, the configuration travels from the leaf to the "test" branch, and then closer to the trunk to the "1.0 release branch." Perforce turns this upside down. A typical development project would create a branch off the tree that is going to server as the working area for the version. When configurations are ready for promotion to test, a branch is created off of the existing branch. The release branch becomes another branch further from the trunk. The benefit of this model is that it supports multiple active maintenance lines, which the typical branching model cannot. For example, suppose that a project wants to split the project after the test stage into two stages to make changes for releases for two different platforms. In the traditional model this is not easy since branches converge as they get closer to release. But in Perforce, branches diverse as they get closer to release, so creating two branches is simple and intuitive. Perforce maintains history of how a configuration moves from branch to branch.

b. File System

Perforce is one of the few tools evaluated that uses the operating system's file system to store its files. Perforce uses the directory structure to provide information about the relevance of a file. For example, if you see a file named "depot/release/1.2/01/hello.c," you know it is part of release 1.2.01. The benefit of this

for high assurance systems is that identifying the product and exporting it to anther system (e.g., for "Trusted CM") is especially straightforward. No trust needs to be placed in the tool's ability to include all the files in the export. One simply copies the appropriate directories to another medium. The metadata can be exported to a SQL database for analysis.

Perforce stores metadata about the files under CM in a database next to the files. The issues related to this are discussed in Section IV.B.1, "Repository Structure: Use Operating System's File System," above.

7. Merant Dimensions

The author gathered information on Merant Dimensions by reviewing product literature on Merant's web site [PVCS03] and through emails with PVCS sales personnel [PVCE03].

Merant Dimensions is one of the most complex, feature-rich CM tools evaluated. It markets itself as the product for the enterprise to manage not just CM, but enterprise-wide process management, issue management, change management, and workflow. Merant Dimensions made a name for itself as PCMS Dimensions when owned by SQL Software. Note that PVCS Professional, also from Merant, has a completely different source and history; it is not just a scaled down version of Dimensions.

Dimension's key features are described below.

a. Lifecycle Management

Dimensions' key concepts in lifecycle management are: lifecycles, states, and change documents. Lifecycles are custom-designed series of activities or states performed in a specific order. ("Off-normal" states such as "failed tests" can be set up for items that cannot be promoted to the next "normal" state.) Entry and exit criteria for each state can be set up to control how configurations move through the states. Change Documents are custom designed electronic forms with up to 220 user-defined fields. Change Documents can be linked to a lifecycle by requiring different parts of the form to be completed by specific people (or specific roles) as part of different states' entry or exit

criteria. Two examples of Change Documents are "Product Change" and "Test Defect Report."

b. Relationships

Dimensions provides a way to link configuration items with other items and with Change Documents. The built-in relationships include "affected by", "information" and "in response to", but users can define additional relationships. In a high assurance project, one might want to link the proof that the FTLS implements the security model with a "proves" relationship. Being able to define relationships between items in a configuration helps to self-document the project, helping external parties become familiar with the project more quickly.

c. Rules

Rules can be created and applied to connect Change Document lifecycles and item lifecycles (or two Change Document lifecycles). There are three types of rules:

- Creation Rules (e.g., must be associated with Change Doc "in response to")
- Action (i.e. cannot move to another state in lifecycle without related Change Doc)
- Closure (i.e. specifies state through which item must pass before it is closed)

d. Access Control

Merant Dimensions allows administrators to assign roles to users against "Design Parts" or parts of your system. Thus, administrators can limit access to parts of the system to those who require access.

e. Export/Import

Dimensions provides several options for importing and exporting files and metadata. Users can export a read-only copy of all the files in a baseline into another system's file system using the "Release" or "Deploy" functions. Metadata can only be exported to another version of Dimensions and must use a Dimensions-created utility. Users can import files from a directory structure into Dimensions. If users want to import

metadata, they can use an XML utility that currently supports conversions from PVCS Professional and Rational ClearCase.

f. Automating the Change Review Process

Merant includes a description of how to use their product to improve a project's Change Review Process. Because of the importance of the Change Review Process in ensuring that only authorized changes are made (Goal 4), the full description below from Merant's "Change Management Capabilities" technical brief [PVCS03] is included below. This process could be used in "Everyday CM" to help ensure that changes made are the appropriate, authorized changes.

> Automating your Change Review Board is an excellent way to take advantage of the power of PVCS Dimensions.
>
> Many development organizations rely on a Change Review Board to control change in their products or system. A Change Review Board (CRB) generally consists of three to five members of the development organization, usually in lead development, project and test management roles. It is the board's job to ensure that all changes entering a system meet the requirements of the customer, have been developed and tested appropriately and do not negatively impact the product or system.
>
> Often, when a change review board is first enacted, the CRB is overwhelmed with the number of changes to a particular product or system. Paper-based systems fail to describe the relationships, dependencies and impacts of a change. Using PVCS Dimensions allows most Change Review Boards to go from a static weekly meeting plus too-frequent emergency sessions to a dynamic online format for approving changes as needed.
>
> **A CRB Scenario**
>
> Karen, the Director of Development for ACME's Customer Service Software System, is ultimately responsible for her customers' satisfaction. She heads up a Change Review Board consisting of three other members:
>
> - Bob - Release Manager
> - Sarah - Test Manager
> - Maya - Business Analyst

The Change Review Board reviews all changes to the system. They actually review and approve each change twice, at the beginning of the change lifecycle and just prior to release. Maya, the business analyst, is responsible for ensuring all changes meet the customer requirements. Bob, the release manager, is responsible to ensure that any change to the system doesn't negatively impact anyone else. The ACME customers do not like to be surprised by down systems, or lost capabilities.

<<author removed the description of the process before the company implemented PVCS Dimensions>>

She bought PVCS Dimensions and implemented an online change review board. Today the Change Review Process looks like this:

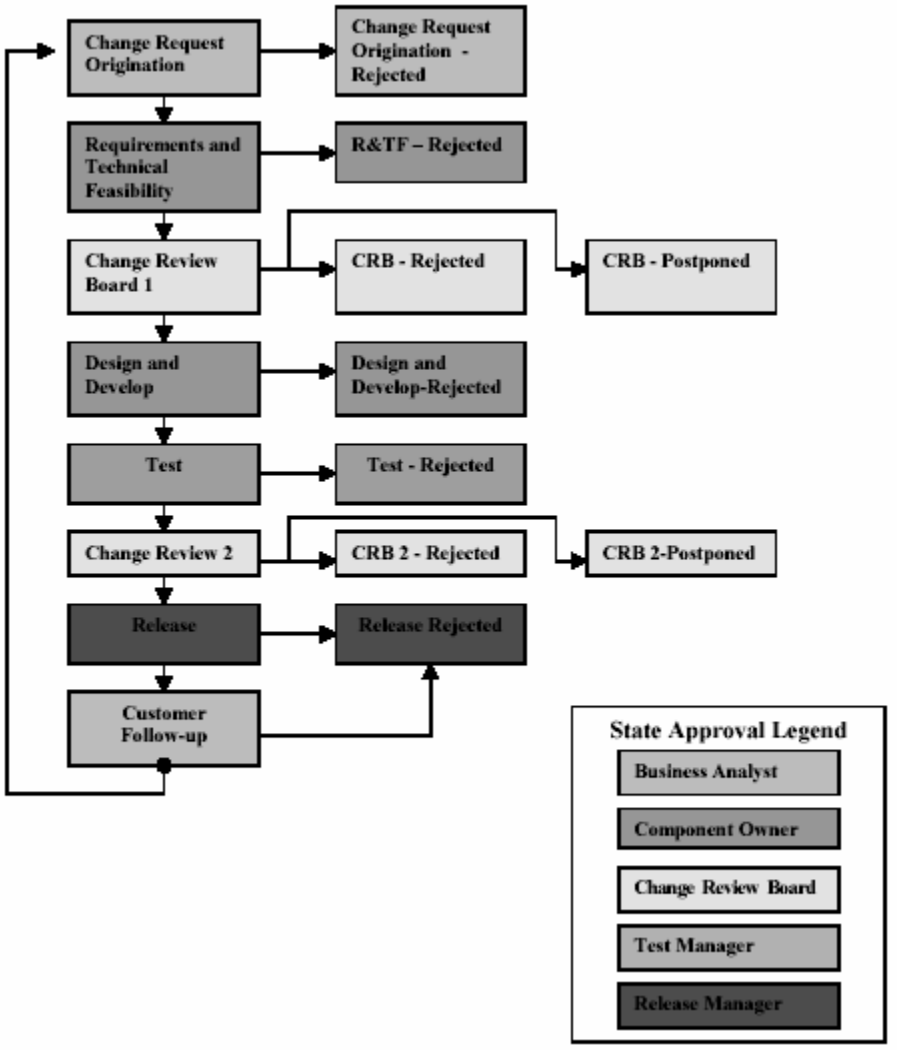

Figure 2. Merant PVCS Dimensions' Change Review Process Example

Change Request State. Maya's customers use the PVCS Dimensions web browser interface (I-NET) to request changes to the system. Maya has added some attributes to the change request that help to filter and clarify the changes. She uses a component attribute to send the change on to the component owner, a lead developer responsible for that aspect of the system. For example, a change request for a new online report shows up in Maya's pending list. The "component attribute" indicates that it is a change to the reports system. Maya reviews the change and actions it to the requirements and technical feasibility state, delegating the change request to Jeff, the lead developer for reports.

Requirements and Technical Feasibility State. Jeff reviews the change for technical feasibility. He works with Maya and the customer to come up with a requirements specification, which he stores in the Dimensions repository and relates to the change request. Jeff also identifies the files to be modified and relates the Change Document to them. He identifies and documents any impacts to the current system, enters a ballpark estimate in the "cost" attribute field, and actions it to the Change Review Board state.

Change Review Board 1 State. Instead of the board meeting each week, each member reviews the changes online. If there are no issues raised after 3 days, Karen, the Development Director, approves the change and actions it back to the component owner, Jeff, for a design and develop phase. An email is automatically sent to the originator of the change request, indicating that the change was reviewed and accepted.

Design and Development State. The Developer makes the required changes, and actions the Change Document and related items to the Sarah, the test manager.

Test State. Sarah tests the change, and if there are no issues, actions the change back to the CRB, where the team reviews it for release, identifying any release impacts to the current system.

Change Review Board 2 State. Bob, the release manager, enters a release date in the "release date" attribute, and Karen, the development director actions the change to the release state.

Release State. The Change Document and related code are released to the system, and Bob actions the Change Document and code to the Customer Follow-up state.

Customer Follow-up State. Maya contacts the customer to ensure they are satisfied with the change. If they are, Maya closes the Change Document. If not, Maya notes the changes and moves the Change Document back to the beginning phase.

Off-Normal States. If there is a problem, the change is moved to an off-normal state. From the off- normal state, a change can be moved to a previous normal state, or it can be sent back to the beginning, depending on the severity of the problem. The Change Review Board has turned on the Change Management Rules, so only the items related to a Change Document can be modified. This has eliminated the number of rogue changes that have entered the system. The Change Review Board still meets on a bi-weekly basis, to discuss medium and long-term projects, and to work out any other process kinks that may have occurred. The CRB realized that there were times when a good change request came through and there weren't enough resources to work the request. Instead of rejecting the request, they've created an off-normal state called "postpone." Used rarely, it holds work that will be addressed at a later date. Every month, Maya runs a Change Document report that shows which change requests are in a postponed state. The CRB reviews the change requests again at the next meeting and may action the change to a normal status as resources become available.

Online Change Reviews and Management improve the quality of the products that you deliver, as well as the timeliness of your delivery. PVCS Dimensions comes complete with out-of-the-box Change Document lifecycles, or, like ACME, you can design one personalized for your environment.

8. StarTeam

The author gathered information on StarTeam from information on Borland's StarTeam website [STWB03], and from email correspondence with Borland's sales personnel [STKS03]. StarBase sold StarTeam until January, 2003, when Borland purchased StarBase. StarTeam is another full-featured tool similar to PVCS. The primary reason it was evaluated in addition to PVCS is its support for threaded discussions.

a. Lifecycle Support

StarTeam, like PVCS Dimensions, has customizable workflow, forms, and process rules. A process rule can, for example, prevent users from adding a file or checking in a change if it does not reference an approved requirement, change request, or

task. When a user does reference the change request, StarTeam automatically creates a link between the file version and change request. Email notification of users can be set up as well.

b. Threaded Discussions

StarTeam provides the ability to have threaded discussions linked to a project, a project folder, or directly to an item within a project. The discussion contents are stored in the same database as the project files. Administrators can determine the types of access users have to the discussions, providing some measure of reputability. These discussions can provide making a permanent record of valuable information on why decisions were made and who made them. The benefits of threaded discussions include:

- A team member can easily incorporate the input from others or ask questions while working on a file.
- Notes explaining why a particular method was or was not used can be included as linked topics.
- Topics can point out things that may have to be changed in a later release of the product.

c. Encryption

StarTeam has an option for strong password enforcement. You can also set different levels of server-based data security using industry standard RSA encryption. StarTeam also supports the locking of all project assets (not just files) to prohibit changes from being overridden by other users.

d. Access Control

Access control functionality is similar to that of PVCS Dimensions

e. Configuration Notions

Each asset is independently versioned, but sets of assets such as files and change requests can be versioned as well, as described in their product literature [STWB03].

> Baselining provides the capability to view a configurable snapshot of the way a project looked at a specific point in time. Each asset in StarTeam is independently versioned. Sets of assets such as files and change requests,

at specific versions, are baselined, natively, inside the system. By comparing baseline views, users can immediately see where volatility, modifications, additions, and deletions have taken

f. Specifications

- **Repository**: Microsoft SQL 7 and 2000, Microsoft Access 2000, IBM® DB2® UDB 7.2, Oracle8i™ (8.1.5, 8.1.6, 8.1.7),Oracle9i™ release 1 (9.0.1.3.0)
- **Server**: Intel® Pentium® 4/900 MHz—1 GHz minimum (Dual or Quad Pentium Intel Xeon® 4/2.26 GHz or higher recommended), 512 MB RAM minimum (1 GB or more recommended), 16 GB or higher hard drive, Caching SCSI controller RAID (minimum 16 MB RAM), 500 MB page file, Microsoft® Windows® 2000 (SP3), Windows NT® Workstation 4.0, or Windows NT Server 4.0 (SP6a), Sun Solaris® 7.0/8.0
- **End User**: Intel Pentium Pro/233 MHz processor, 256–512 MB; RAM; 300 MB page file; Microsoft Windows 98®, Windows NT 4.0, Windows 2000, Windows XP®, or Java™ enabled Linux® and Unix operating systems.

LIST OF REFERENCES

ACRV03 AccuRev Product Information [http://www.AccuRev.com/product], April 2003.

ANDR02 Anderson, E.A., *A Demonstration Of The Subversion Threat: Facing A Critical Responsibility In The Defense Of Cyberspace*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 2002.

ARBM03 Telephone conversation between Bob Manning, AccuRev and author, 1 May 2003.

ARDC03 Telephone conversation and wb demonstration between David Crawford and Bob Manning, AccuRev, and author, 23 April 2003.

BERC03 Berczuk, S. with Appleton, B., *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, Addison-Wesley, 2003

BERL95 Berlack, H. Configuration Management International

BKWB03 BitKeeper Product Information [http://www.bitkeeper.com], May 2003.

BKLX03 Bar, Joe, "Larry McVoy on BitKeeper, kernel development, Linus Torvalds & Bruce Perens" [http://www.linuxworld.com/site-stories/2003/0127.barr.html], originally published 27 Jan 2003.

BURR99 Burrows, C., "Configuration Management: Coming of Age in the Year 2000" [http://www.stsc.hill.af.mil/crosstalk/1999/03/burrows.asp], originally published March 1999, accessed June 2003.

CISR02 Irvine, C., Levin, T., Dinolt G., *Diamond High Assurance Security Program: Trusted Computing Exemplar,* White Paper, Center for Information Systems Security Studies and Research, Naval Postgraduate School, Monterey, CA, September, 2002. http://cisr.nps.navy.mil/downloads/Project_TCExemp2.pdf.

CLCS03 ClearCase Product information [http://www.rational.com/products/clearcase/], May 2003.

CLJS03 Interview between John Sovereign, User of Rational ClearCase, and author, 20 April 2003.

CMMS95 Carnegie Mellon University, Software Engineering Institute, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, 1995.

CMTS88 National Computer Security Center, *A Guide to Understanding Configuration Management in Trusted Systems*, Fort Meade, MD, 28 March 1988.

COMC99 *Common Criteria*, v. 2.1, part 3, August 1999.

CVSS03 Subversion Open Source Group, "Features Planned for Subversion 1.0," [http://subversion.tigris.org/], May 2003.

CVSD03 Interview between George Dinolt, user of CVS, and author, May 2003.

CVSF99 Fogel, K., *Open Source Development with CVS*, Coriolis, 1999.

CVSC03 Cederqvist, P., "Version Management with CVS" [http://www.cvshome.org/docs/manual/cvs-1.11.6/cvs.html], Accessed June 2003.

CVSW03 CVS Project Website [http://www.cvshome.org], May 2003.

DART00 Dart, S., *Configuration Management: The Missing Link in Web Engineering,* Artech House, 2000.

DODD02 DoD Directive 8500.1, "Information Assurance (IA)," 24 October 2002.

DODD85 DoD 5200.28-STD, "DoD Trusted Computer Security Evaluation Criteria," December 1985.

EVAL03 National Computer Security Center, "Historical Evaluated Product List" [http://www.radium.ncsc.mil/tpep/epl/historical.html], accessed June 2003.

FOAS03 Federation of American Scientists, "Strategic Automated Command Control System" [http://www.fas.org/nuke/guide/usa/c3i/saccs.htm], May 2003.

GTNP95 National Computer Security Center, "Final Evaluation Report: Gemini Trusted Network Processor Version 1.02," Gemini Computers, Incorporated, 28 June 1995.

IRCM02 Clemm, G., and others, "Impact of the Research Community on the Field of Software Configuration Management: Summary of an Impact Project Report," ACM SIGSOFT Software Engineering Notes, v. 27, no 5, pp.31-39, September 2002.

LEON00 Leon, A., *A Guide to Software Configuration Management*, Artech House, 2000.

OPAI02 Shapiro, J.S. and Vanderburgh, J., ``Access and Integrity Control in a Public-Access, High-Assurance Configuration Management System," Proc. 11th USENIX Security Symposium, 2002, San Francisco, CA, 2002.

OPCN02 Shapiro, J.S. and Vanderburgh, J., “CPCMS: A Configuration Management System Based on Cryptographic Names,” Proc. 2002 USENIX Annual Technical Conference, FreeNIX Track, Monterey, CA, 2002.

OPEL03 Shapiro, J.S., Vanderburgh, J., and Lloyd, J., ``OpenCM: Early Experiences and Lessons Learned," Proc. 2003 USENIX Annual Technical Conference, FreeNIX Track, San Antonio, Texas, 2003.

OPEN03 The EROS Group, LLC and Johns Hopkins University, “OpenCM User’s Guide,” [http://www.opencm.org/opencm.html], April 2003.

OPJL03 Lloyd, J., Email message, Subject: RE: Dinolt - NPS Thesis re: OpenCM, 13 June 2003.

OPJS03 Personal communication between Jonathan S. Shapiro and author, 12/13 June 2003.

OPJV03 Vanderburgh, J., Email message, Subject: RE: Dinolt - NPS Thesis re: OpenCM, 13 June 2003.

ORNG85 National Computer Security Center, “DoD Trusted Computer System Evaluation Criteria,” Fort Meade, MD, 1985.

PFWB03 Perforce Product Information [http://www.perforce.com/], May 2003.

PFWG03 Interview between Cynthia Walston and Kate Gentry, Users of Perforce, and the author, 16 May 2003.

PKCR98 Netscape®, “Introduction to Public-Key Cryptography” [http://developer.netscape.com/docs/manuals/security/pkin/contents.htm], last updated 9 October 1998, accessed June 2003.

PVCE03 Meshell, J., Email message, Subject: RE: Navy Wide Enterprise License For PVCS, 4 June 2003.

PVCS03 Merant Dimensions Product Information [http://www.merant.com/Products/ECM/dimensions/], May 2003.

RADL03 Personal communication between George Dinolt and Rance DeLong, one of the KSOS developers. May 2003.

RTCA03 RTCA, Inc., “Document Descriptions,” [http://www.rtca.org/downloads/documentdescriptions.pdf], May 2003.

STKS03 Smith, K., Email message, Subject: Questions re:StarTeam for Naval Postgraduate School, 2 June 2003.

STWB03 StarTeam Product Information [http://www.borland.com/starteam/], May 2003

TCAS99 Abdul-Baki, B., Baldwin, J. and Rudel, M. “Independent Validation And Verification Of The TCAS Collision Avoidance Subsystem,” AIAA 18th Annual Digital Avionics Systems Conference, 1999.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. Ernest McDuffie
   National Science Foundation
   Arlington, VA

4. RADM Zelebor
   N6/Deputy DON CIO
   Arlington, VA

5. Russell Jones
   N641
   Arlington, VA

6. David Wirth
   N641
   Arlington, VA

7. CAPT Sheila McCoy
   Headquarters U.S. Navy
   Arlington, VA

8. CAPT Robert Zellmann
   CNO Staff N61
   Arlington, VA

9. Dr. Ralph Wachter
   ONR
   Arlington, VA

10. Dr. Frank Deckelman
    ONR
    Arlington, VA

11. Richard Hale
    DISA
    Falls Church, VA

12. George Bieber
OSD
Washington, DC

13. Deborah Cooper
DC Associates, LLC
Roslyn, VA

14. David Ladd
Microsoft Corporation
Redmond, WA

15. Marshall Potter
Federal Aviation Administration
Washington, DC

16. Ernest Lucier
Federal Aviation Administration
Washington, DC

17. Keith Schwalm
DHS
Washington, DC

18. RADM Joseph Burns
Fort George Meade, MD

19. Howard Andrews
CFFC
Norfolk, VA

20. Steve LaFountain
NSA
Fort Meade, MD

21. Penny Lehtola
NSA
Fort Meade, MD

22. Dr. George Dinolt
Computer Science Department
Naval Postgraduate School
Monterey, CA

23. Michael Thompson
Computer Science Department
Naval Postgraduate School
Monterey, CA

24. Lynzi Ziegenhagen
Civilian
Naval Postgraduate School
Monterey, CA 93943